Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Comunicações

DESENVOLVIMENTO DE UM DECODIFICADOR DE ÁUDIO EMBARCADO PARA O ISDB-TB

Autor: Vinícius José Andrade Braga

Orientador: Prof. Dr. Luís Geraldo P. Meloni

Dissertação de Mestradoapresentada à Faculdade de Engenharia Elétrica e de Computaçãocomo parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: **Telecomunicações e Telemática.**

Banca Examinadora

Prof. Dr. Luís Geraldo P. Meloni (presidente) — DECOM/FEEC/UNICAMP
Prof. Dr. Osamu Saotome — IEE/ITA
Prof. Dr. Yuzo Iano — DECOM/FEEC/UNICAMP

Campinas – SP
Abril de 2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

B73d

Braga, Vinícius José Andrade
Desenvolvimento de um decodificador de áudio embarcado para o ISDB-Tb / Vinícius José Andrade Braga. --Campinas, SP: [s.n.], 2011.

Orientador: Luís Geraldo Pedroso Meloni.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Processamento de sinais - Tecnicas digitais. 2. Sistemas embutidos de computador. 3. Televisão digital. 4. Otimização. I. Meloni, Luís Geraldo Pedroso. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Development of an embedded audio decoder for ISDB-Tb
Palavras-chave em Inglês: Digital signal processing, Embedded computer systems, Digital television, Otimization
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Osamu Saotome, Yuzo Iano
Data da defesa: 20-04-2011
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Vinicius José Andrade Braga

Data da Defesa: 20 de abril de 2011

Título da Tese: "Desenvolvimento de um Decodificador de Áudio Embarcado para ISDB-Tb"

Prof. Dr. Luis Geraldo Pedroso Meloni (Presidente): _____

Prof. Dr. Osamu Saotome: _____

Prof. Dr. Yuzo Iano: _____

Resumo

Este trabalho descreve o desenvolvimento de um decodificador de áudio embarcado em um *Digital Signal Processor* (DSP)de acordo com o padrão *High Efficiency* AAC *version* 2(HE-AAC v2) do MPEG-4. Essa atividade é parte integrante do projeto Rede H.264 que tem por objetivo o desenvolvimento de tecnologias nacionais para ser integrado ao padrão brasileiro de TV digital, o *Integrated Services Digital Broadcasting-Terrestrial Brazilian version*(ISDB-Tb). Também apresenta um estudo sobre diversas técnicas de otimização para processamento em tempo real na busca de se obter o melhor desempenho da arquitetura utilizada. Como resultado final deste trabalho, chegou-se a um decodificador embarcado em tempo real, otimizado com as técnicas descritas e compatível com o ISDB-Tb.

Palavras-chave: decodificador de áudio, ISDB-Tb, MPEG-4, sistemas embarcados, DSP.

Abstract

This work describes the development of an embedded audio decoder in a Digital Signal Processor (DSP) according to the standard High Efficiency AAC v2 (HE-AAC v2) of MPEG-4. This activity is part of the Rede H.264 project which has objective the development of national technologies to be integrated in the Brazilian Digital TV standard, the Integrated Services Digital Broadcasting-Terrestrial Brazilian version (ISDB-Tb). It also presents a study of various optimization techniques for real-time processing in the quest to get the best performance of the architecture used. As final result of this work a real-time embedded decoder was achieved, optimized with the techniques described and compatible with the ISDB-Tb.

Keywords: audio decoder, ISDB-Tb, MPEG-4, embedded systems, DSP.

Agradecimentos

São tantos os agradecimentos e tantas as pessoas a serem agradecidas pela realização deste trabalho que certamente não seria possível citar todos aqui.

Mas quero dedicar um agradecimento especial ao Pai, pois é dEle, toda honra e toda glória alcançada em minha vida.

Agradeço também à minha família, que me fez ser o homem que sou, me ensinando a importância de Deus e da instituição da família na formação de um cidadão e de um profissional íntegro.

A todos do laboratório RT-DSP por toda a paciência, disponibilidade, atenção e companheirismo, que foram imprescindíveis para que alcançasse esta vitória.

A meus amigos, colegas, companheiros de viagem, companheiros de boteco que estiveram ao meu lado durante estes 988 dias e que sem eles, não teria conseguido chegar até aqui.

Obrigado de coração a todos vocês e tenham certeza que poderão contar comigo quando e para o que precisarem.

Lista de Ilustrações

Lista de Tabelas

Lista de Siglas

3GPP – *Third Generation Partnership Project*

AAC – *Advanced Audio Coding*

AD – Analog Devices

ALU – *Arithmetic Logic Unit*

AU – *Access Unit*

CAM – *Content Addressable Memory*

CCLK – *Core Clock*

CLKIN – *Input Clock*

CPLB – *Cacheability Protection Lookaside Buffer*

DAG – *Data Address Generators*

DCT – *DiscreteCosine Transform*

DMA – *Direct Memory Access*

DSP –*Digital Signal Processor*

EBIU – *External Bus Interface Unit*

EPROM – *Erasable Programmable Read Only Memory*

ETSI –*EuropeanTelecommunications Standards Institute*

FFT –Transformada Rápida de Fourier

FIFO – *First in, First out*

HE-AAC –*High Efficiency Advanced Audio Coding*

IDDE – *Integrated Development and Debugger Environment*

IMDCT – *Inverse Modified Discrete Cosine Transform*

ISDB-T – *Integrated Services Digital Broadcasting-Terrestrial*

ISDB-Tb –*Integrated Services Digital Broadcasting-Terrestrial Brazilian version*

KBD – Janela derivada de Kaiser-Bessel

LC – *Low Complexity*

LDF – *Linker Description File*

LRU – *Least Recently Used*

LTP –*LongTerm Prediction*

M/S – *Mid/Side*

MAC – Multiplicador e Acumulador

MDCT – *Modified Discrete Cosine Transform*

MMU – *Memory Management Unit*

MP3 –MPEG-1 *Layer* III

MPEG – *Moving Pictures Expert Group*

PAM – *Psycho-Acoustic Model*

PCI – *Peripheral Component Interconnect*

PCM – Modulação por Código de Pulsos

PLL – *Phase Locked Loop*

PPI – *Parallel Peripheral Interface*

PQF – *Polyphase Quadrature Filter*

PS – *Parametric Stereo*

QMF – *Quadrature Mirror Filtering*

Q-Loop – *Quantization Loop*

ROM – *Random Only Memory*

RISC – *Reduced Instruction Set Computing*

RT-DSP – Laboratório de Processamento de Sinais em Tempo Real

SBR – *Spectral Band Replication*

SBTVD – Sistema Brasileiro de Televisão Digital

SCLK – *System Clock*

SDRAM – *Synchronous Dynamic Random Access Memory*

SMR – Relação Sinal-Máscara

SNR – Relação Sinal-Ruído

SPI – *Serial Peripheral Interface*

SPORTs – *Serial Ports*

SPP –*Spectrum Process*

SRAM – *Static Random Access Memory*

SSR – *Scalable Sampling Rate*

TDAC – *Time-Domain Aliasing Cancellation*

TNS – *Temporal Noise Shaping*

TwinVQ –*Transformation-domain Weighted Interleave Vector Quantization*

UART – *Universal Asynchronous Receiver Transmitter*

USB – *Universal Serial Bus*

VCO – *Voltage Controlled Oscillator*

Sumário

Introdução

Desde o final da década de 1990, o Brasil vem desenvolvendo testes, avaliações e pesquisas para a definição de um padrão de transmissão de televisão digital.Uma das expectativas era proporcionar uma transmissão e recepção de sinais com maior qualidade do conteúdo podendo obter imagens em alta definição. Além disso, o desenvolvimento de um sistema digital também tinha um cunho social, pois se pretendia usá-lo para realizar a inclusão digital para grande parcela da população que não tem acesso às tecnologias de comunicação.

Depois de diversos estudos, o Brasil optou por desenvolver um padrão brasileiro baseado no padrão *Integrated Services Digital Broadcasting-Terrestrial*(ISDB-T) usado no Japão. Esse padrão brasileiro ficou conhecido como Sistema Brasileiro de Televisão Digital (SBTVD) ou *Integrated Services Digital Broadcasting-Terrestrial Brazilian version* (ISDB-Tb). As principais diferenças entre estes padrões são o emprego de um *middleware* desenvolvido especificamente para o novo padrão brasileiro e a substituição do padrão de codificação de áudio e vídeo MPEG-2 pelo MPEG-4. Essas melhorias fazem do ISDB-Tb, um dos padrões de TV digital mais eficientes disponíveis no mercado.

Visando promover o SBTVD,o Ministério de Ciência e Tecnologia criou uma rede de pesquisa em convênio junto à FINEP, denominada “Rede H.264 SBTVD”, formada por diversos laboratórios em diversas universidades do país. Dentre eles, coube ao Laboratório de Processamento de Sinais em Tempo Real (RT-DSP)da UNICAMP, o desenvolvimento de um decodificador de áudio em *hardware* de acordo com o padrão *High Efficiency* AAC *version* 2 (HE-AAC v2), que faz parte das *suites* de especificações do MPEG-4.

Nesse sentido, esta dissertação apresenta o processo de desenvolvimento desse decodificador de áudio em um *Digital Signal Processor* (DSP). Além disso, também é exposto um estudo detalhado sobre técnicas de otimização e suas aplicações nesse decodificador com a finalidade de que esse opere com maior eficiência quanto à complexidade computacional.

As principais contribuições deste trabalho referem-se a:

- Definir uma referência algorítmica para o decodificador de áudio que atenda as especificações do SBTVDvoltadas a um DSP;
- Um estudo detalhado de diversas técnicas de otimização voltadas para aplicações

embarcadas em DSPs.

- Aplicação dessas técnicas estudadas em implementação embarcada, visando maior eficiência quanto à complexidade computacional.

O primeiro capítulo apresenta a evolução dos padrões de codificação de áudio da família *Moving Picture Experts Group* (MPEG)relatando suas principais características até chegar ao padrão *Advanced Audio Coding* (AAC), conforme padronizado no MPEG-4. Também nesse capítulo,sãomostradas as novas funcionalidades apresentadas nessa versão do padrão com destaque para as ferramentas de *software* denominadas *Spectral Band Replication* (SBR) e para o *Parametric Stereo* (PS).

Em seguida, no capítulo 2, são exibidos, de forma detalhada, todos os fundamentos envolvidos no HE-AAC v2, apresentando cada ferramenta, sua funcionalidade e os ganhos proporcionados por cada uma delas.

No capítulo 3, é feita uma abordagem sobre o dispositivo de *hardware* que é utilizado para o desenvolvimento desse projeto, descrevendo suas principais características, seus principais recursos e a forma como é feita a comunicação entre o computador e o DSP escolhido.

Posteriormente, no capítulo 4, tem-se início a descrição do processo de otimização realizado. Com o intuito de explicar e justificar as técnicas relacionadas nesse capítulo no desenvolvimento deste trabalho, cada uma delas é descrita, explicada e ainda é apresentada uma comparação dos ganhos obtidos com sua utilização em funções ou trechos do próprio programa.

No capítulo 5, são expostos, além de um resumo das etapas dodesenvolvimento deste trabalho, os resultados finais alcançados com a aplicação das técnicas descritas no capítulo anterior. Por fim, nesse capítulo também é realizada uma comparação do produto final desenvolvido com uma implementação semelhante disponível na literatura técnica especializada.

Capítulo 1

Breve Introdução aos Padrões de Áudio da Família MPEG

Desde o surgimento dos discos compactos(CD) em 1982, o áudio digital se tornou cada vez mais presente no dia-a-dia. Nesse momento, em um único CD, era possível o armazenamento de mais de 60 minutos de áudio de alta qualidade. No início isso era excelente, mas com o surgimento da internet e a com a sua popularização houve a necessidade de se criar mecanismos de compressão ainda maiores diante das limitações de largura de banda da rede.

Diante disso, alguns grupos se reuniram com o intuito de estabelecer padrõespara essa codificação visando um melhor aproveitamento das novas tecnologias aliadas à alta qualidade. Um dos primeiros e mais importantes grupos criados com a finalidade de estabelecer um padrão foi o *Moving Picture Experts Group* (MPEG), que em 1988 padronizou a compressão/descompressão, o processamento e a representação codificada de áudio, vídeo e da combinação desses. Surgia assim o MPEG-1.

Uma das partes desse padrão trata especificamente da codificação de áudio [1], na qual são descritas a sintaxe e a semântica para três classes de métodos de compressão conhecidos como*layers.*O mais conhecido deles é o MPEG-1 *Layer*III (MP3), o qual desde seu surgimento tornou-se o principal padrão e que até hoje está presente em muitos dispositivos.

Algumas das características que fizeram do MP3 um padrão tão popular foi a utilizaçãode uma variante da TransformadaCosseno Discreta (DCT), conhecida como *Modified Discrete Cosine Transform*(MDCT) e a adoção de modelos psico-acústicos para a redução da taxa de dados necessária para um fluxo de áudio.

Alguns anos mais tarde, o MPEG se reuniu novamente com a finalidade de aperfeiçoar o padrão já existente. Nesse encontro, em 1994, foi criado o MPEG-2 (ISO/IEC 13818) que trouxe uma versão do MP3 com algumas melhorias, entre elas, a codificação multicanais com até 5.1 canais e a implementação do *Advanced Audio Coding*(AAC)[2].

Em 1998, em mais um encontro do mesmo grupo, foi criado um novo padrão, o MPEG-4 (ISO/IEC 14496) o qual priorizou a inclusão de novas funcionalidades, como o *Temporal Noise Shaping* (TNS), o *Long Term Prediction* (LTP) e o *Transformation-domain Weighted Interleave VectorQuantization*(TwinVQ), ao invés de aumentar a eficiência da compressão.

Posteriormente o padrão AAC foi novamente aprimorado com a introdução de duas novas tecnologias. Em 2003 com a introdução do *Spectral Band Replication* (SBR) surgiu o *High Efficiency* AAC (HE-AAC), e em 2006 foi introduzido o *Parametric Stereo*(PS)criando assim o HE-AAC v2.O SBR estipula que a região de alta frequência de um sinal após passar por um filtro passa-baixa pode ser recuperado com base nesse sinal existente e uma pequena quantidade de dados de controle. Já o PS aumenta a eficiência da codificação estéreo realizando codificações esparsas no domínio do tempo de forma semelhante ao que o SBR faz no domínio da frequência.

O grupo MPEG não foi o único a buscar um padrão de codificação de áudio. Algumas empresas também criaram seus padrões de acordo com suas necessidades sendo válido citar dentre eles o DolbyAC-2/3desenvolvido pela Dolby Digital e o SonyATRACdesenvolvido pela Sony. A seguir naTabela 1.1,é apresentado um quadro comparativo entre os principais padrões de codificação de áudio.

Tabela 1.1–Algumas características dos principais *codecs* de áudio disponíveis [3].

Codec	Taxa para qualidade (kb/s)	Principais Aplicações
MPEG-1 *Layer* I	192 por canal de áudio estéreo	Cassete Compacto Digital
MPEG-1 *Layer* II	128 por canal de áudio estéreo	DAB, CD-I, DVD
MPEG-1 *Layer* III	96 por canal de áudio estéreo	ISDN, Sistemas de Rádio via Satélite, Áudio de Internet
Dolby AC-2	128por canal de áudio estéreo	Ponto a Ponto, Cabo
Dolby AC-3	384 para os 6 canais de áudio	Ponto à Multiponto, HDTV, cabo, DVD, Cinema, LaserDisc
Sony ATRAC	140 por canal	MiniDisc
MPEG-2 AAC	384 para os 6 canais de áudio	HDTV, DVD, Rádio na Internet

1.1 O SISTEMA DE CODIFICAÇÃO MPEG-4 AAC

Tanto o AAC quanto o HE-AAC v2 tem uma estrutura semelhante àmostrada naFigura 1.1. Sua estrutura básica é composta dos seguintes módulos: *Psychoacoustic Model* (PAM), MDCT, *Spectrum Process*(SPP) e *Quantizantion Loop* (Q-Loop).

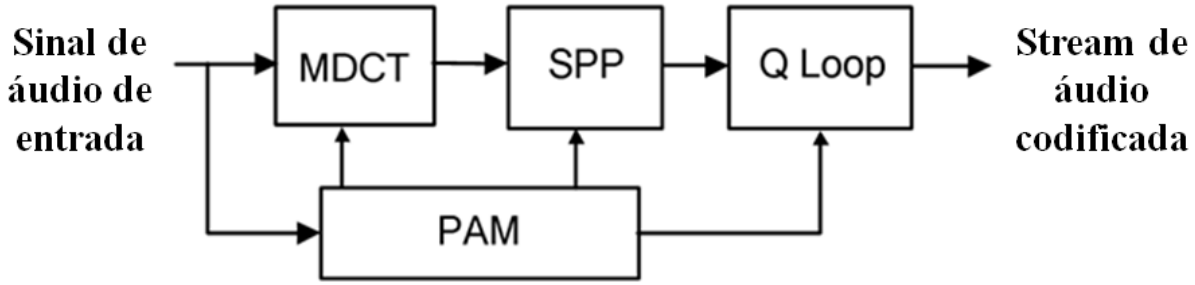


Figura 1.1 - Bloco básico do codificador MPEG-2/4 AAC[3].

Segundo [3] a MDCT transforma as amostras do sinal de entrada que estão no domínio do tempo para um domínioemfrequência não o de Fourier. Ao mesmo tempo o PAM calcula a relação sinal-máscara (SMR) que é usada para determinar a precisão do Q-Loop. A saída do PAM também inclui a informação do tipo de bloco que será utilizado pela MDCT.

Depois de a MDCT converter os dados em espectros, os coeficientes da MDCT são transferidos para o SPP que é usado para remover suas redundâncias e irrelevâncias através da codificação *Joint Stereo* e do TNS (*Temporal Noise Shaping*). Finalmente, os espectros realizam a quantização não-uniforme e a codificação sem ruído baseado no limiar de mascaramento e no número disponível de *bits* para minimizar o erro de quantização audível no Q-Loop.

1.1.1 Esquema de Compressão HE-AAC v1

Como foi dito anteriormente, a introdução da nova tecnologia SBR ao AAC deu origem ao HE-AAC v1.A ideia básica por trás do SBR é a observação de que normalmente existe uma forte correlação entre as características da faixa de alta frequência de um sinal e as características da faixa de baixa frequência do mesmo sinal. Assim, uma boa aproximação para a representação das altas frequências do sinal de entrada pode ser alcançada pela transposição das baixas frequências, cujo processoé ilustrado na Figura 1.2.

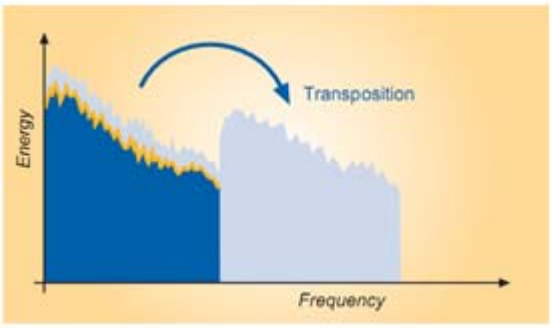


Figura 1.2 - Criação de altas frequências pela transposição.Reproduzido de [4]

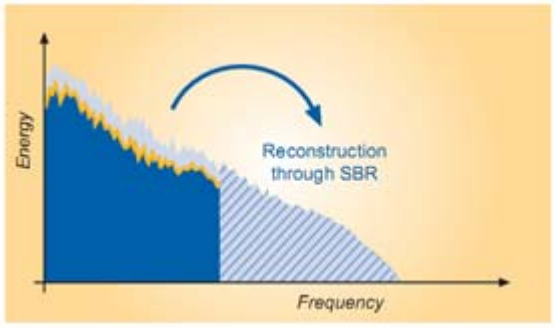


Figura 1.3 - Ajuste da envoltória espectral da banda alta. Reproduzido de [4].

Além da transposição, a reconstrução das altas frequências, como pode ser visto na Figura 1.3, é realizada pela envoltória espectral do sinal de entrada original ou por meio de informações adicionais para compensar uma falta potencial de componentes de alta frequência.

Além do SBR, o HE-AAC v1 adicionou algumas ferramentas úteis ao núcleo do codificador AACcomo o banco de análise *Quadrature Mirror Filterbank*(QMF),o cálculo da envoltória de dados e o*Down-Sampler*. A Figura 1.4mostra o codificador com essas novas ferramentas.

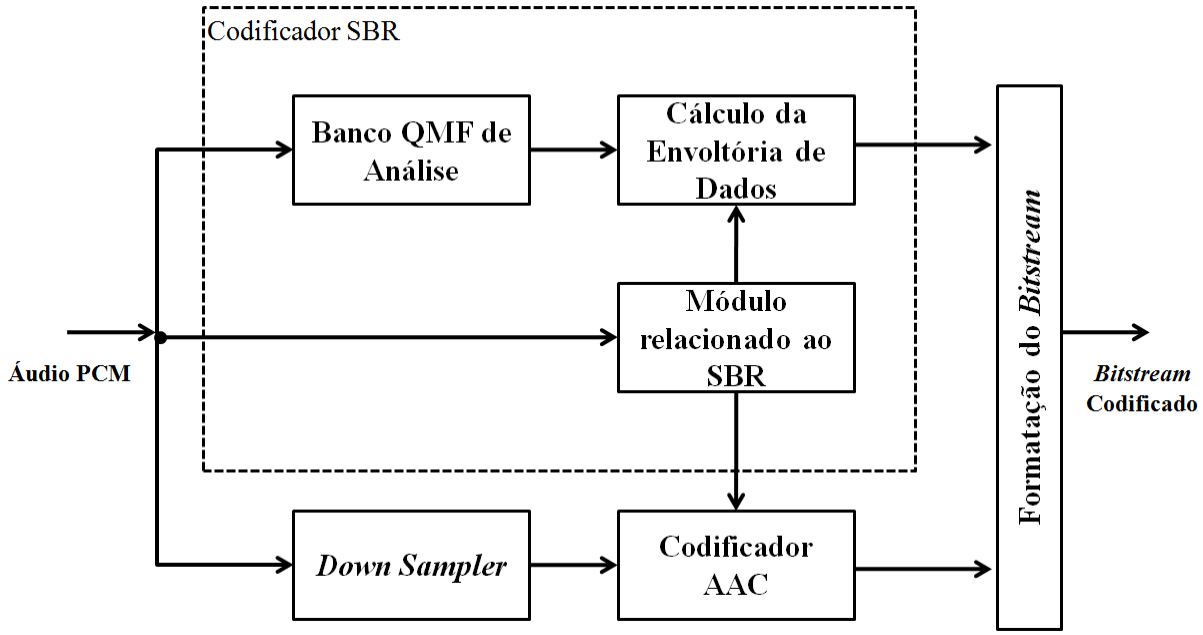


Figura 1.4 - Diagrama de blocos do codificador MPEG-4 HE-AAC v1[3].

Todas as taxas de amostragem do sinal são empregadas para a configuração do codificador SBR e para o *Down*-Sampler diretamente. Os sinais da modulação por código de pulsos (PCM)com a metade da taxa de amostragem que são a alimentação do codificador AAC são dizimados pelo*Down-Sampler*. Ao codificador SBR, fica o encargo de estimar os parâmetros de controle para garantir que o resultado da reconstrução das altas frequências seja o mais semelhante possível com o sinal original.

1.1.2Esquema de Compressão HE-AAC v2

O desenvolvimento do PSaperfeiçoou novamente o padrão de codificação de áudio, resultando no HE-AAC v2. O princípio do PS é a transmissão de um sinal mono codificado em conformidade com o formato HE-AAC em conjunto com a descrição de uma imagem paramétrica para formar o estéreo. A Figura 1.5ilustra esse princípio.

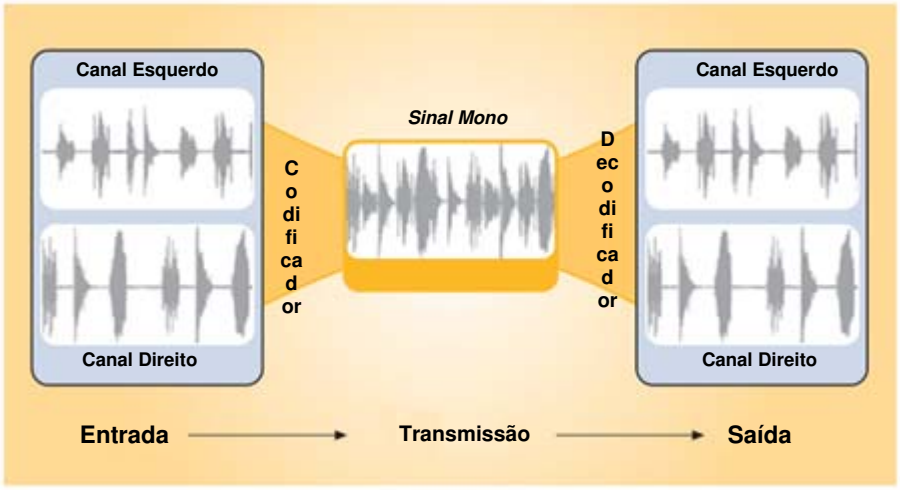


Figura 1.5 - Princípio básico do processo de codificação do*Parametric Stereo*[4].

A grosso modo, o AAC é usado na codificação das baixas frequências, o SBR codifica as altas frequências e o PS codifica a imagem estéreo de forma parametrizada. O diagrama de blocos do codificador HE-AAC v2 é mostrado naFigura 1.6.

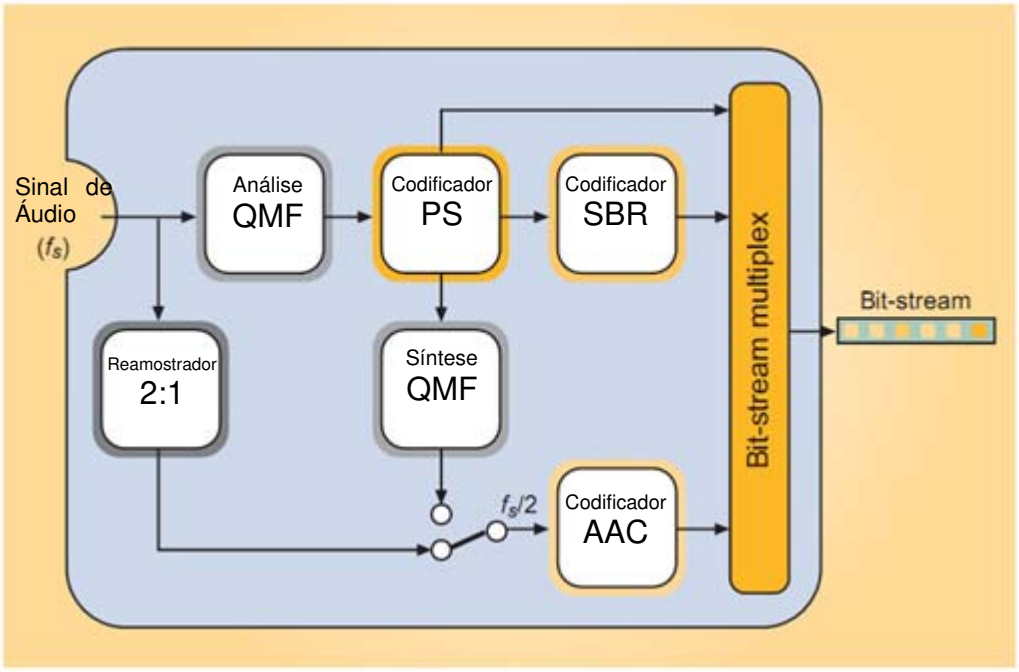


Figura 1.6 - Diagrama de blocos do codificador HE-AAC v2 [4].

De acordo com [4],se a ferramenta PS for usada, o codificador PS vai extrairas informações necessárias baseadas no resultado das amostras do filtro de análise QMF e, em seguida, aplica o *downmix* no sinal estéreo para transformá-lo em uma versão mono. Com um filtro de síntese QMF essa representação mono vai ser transformada novamente para o domínio do tempo e introduzida no codificador AAC.Do contrário, caso o PS não seja utilizado, o sinal de entrada é re-amostrado na proporção de 2:1 e, novamente, o sinal dizimado alimenta o codificador AAC.

Conforme já citado, o HE-AAC v2 foi escolhido para ser a tecnologia de codificação de áudio a ser utilizada no SBTVD, por ser um dos mais modernos codificadores de áudio da atualidade.

Capítulo 2

Os Fundamentos do Codificador MPEG-4 AAC

Este capítulo apresenta uma descrição mais detalhada de cada uma das ferramentas que compõem a estrutura de um codificador de áudio AAC do MPEG-4. Conforme já mencionado no texto, existem algumas diferenças entre o AAC presente no MPEG-2 e no MPEG-4, mas ambos possuem a mesma espinha-dorsal. As principais alterações no codificador AAC do MPEG-4 resumem-seà utilização das novas ferramentas citadas no capítulo anterior. A Figura 2.1mostra a estrutura do HE-AAC v2, no entanto nem todas as ferramentas encontram-se representadas no diagrama de blocos da figura em questão, pelo fato dessas ferramentas não serem utilizadas nesta implementação. Optou-se por enfatizar somente as ferramentas que são utilizadas para atender as necessidades da norma brasileira do SBTVD.

2.1 PERFIS DE ÁUDIO

Outra diferença entre os padrões de áudio na suíte do MPEG-2 e MPEG-4 está nos perfis definidos para cada um. No AAC do MPEG-2 são oferecidos três diferentes perfis [2]:

1- *Main Profile*;
2- *Low complexity Profile* (LC);
3- *Scalable Sampling Rate Profile* (SSR).

Já o AAC do MPEG-4 apresenta diversos perfis, sendo que os relacionados ao escopo deste trabalho são os seguintes[5]:

1- AAC *Profile* – contém o objeto AAC-LC, que é a contrapartida do MPEG-2 AAC LC *Profile*;
2- *High Efficiency* AAC *Profile* – contém o objeto SBR e o AAC-LC;
3- *High Efficiency* AAC *Version* 2 *Profile* – representa todos os objetos presentes no HE *Profile* combinados com a ferramenta PS.

Como pode ser observado, cada um desses perfis relata um nível de qualidade e

complexidade para um determinado tipo de áudio. A Tabela 2.1apresenta o identificador dos objetos de áudio para os diferentes perfis.

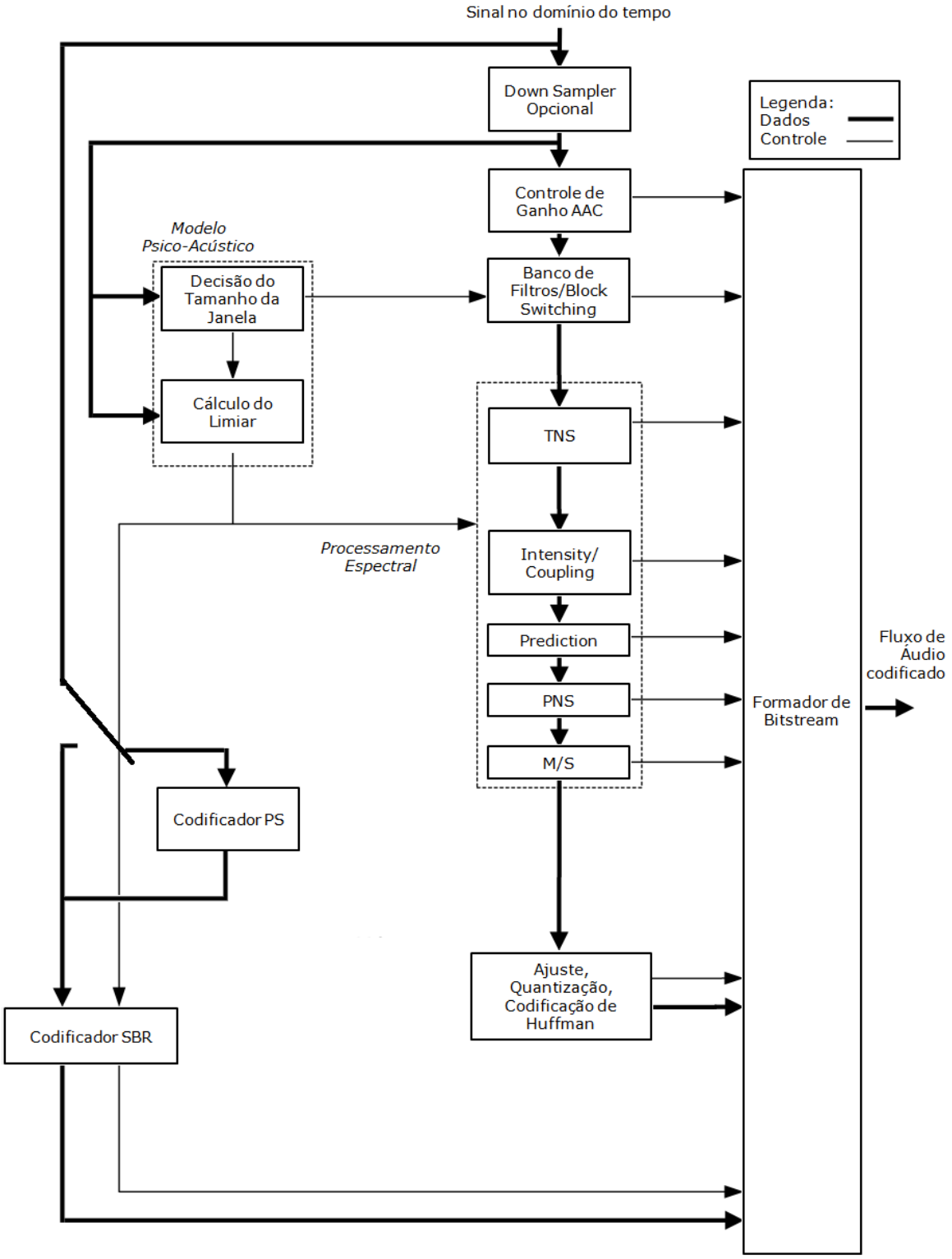


Figura 2.1 - Diagrama de blocos do codificador HE-AAC na versão 2[5].

Tabela 2.1 - Definição dos perfis de áudio do MPEG-4[5].

ID do Tipo de Objeto	Tipo de Objeto de Áudio	AAC *Profile*	*High Efficiency* AAC *Profile*	*High Efficiency* AAC v2 *Profile*
2	AAC LC	x	x	x
5	SBR		x	x
28	PS			x

2.2 FERRAMENTAS DO CODIFICADOR HE-AAC v2

Para fins de apresentação de uma breve explanação da funcionalidade de cada ferramenta, será tomado como referência o perfil de áudioHE-AACv.2, por ser o mais abrangente e por incluir as funcionalidades dos demais perfis já citados.

2.2.1Modelo Psico-acústico

O modelo psico-acústico (PAM) é uma das principais ferramentas na codificação de áudio perceptual. O princípio dessa ferramenta baseia-se no fato de que o sistema auditivo humano não é sensível a todas as frequências de uma fonte sonora de uma mesma forma.Por isso, é necessário que se descreva um modelo que simule o ouvido humano com todas suas limitações e assim possa descartar informações irrelevantes presentes no sinal.

O PAM é um modelo matemático que tenta simular da melhor forma possível as capacidades da audição humana, ou seja, a sensibilidade do ouvido humano em relação à intensidade sonora, à seletividade espectral e o efeito de mascaramento. A composição desse modelo é baseada em diversos estudos relacionados à percepção humana.

No decorrer dos anos, o MPEG desenvolveu dois modelos psico-acústicos para serem utilizados em seus padrões de codificação. O modelo psico-acústico utilizado no AAC é o PAM II. Ele tem um bom desempenho para a simulação do sistema humano, mas possui um alto custo computacional se comparado ao PAM I, seu antecessor. O PAM é aplicado somente na etapa de codificação do sinal, o que permite reduzir a complexidade do decodificador.

A Figura 2.2 mostra o diagrama de blocos do PAM. Em linhas gerais,esse modelo tem duas tarefas principais no codificador MPEG: decidir qual o tipo de bloco que vai ser usado no banco de filtros e calcular a SMR para cada sub-banda. De acordo com[3], o modelo psico-

acústico é dividido em 13 passos principais, os quais são descritos a seguir:

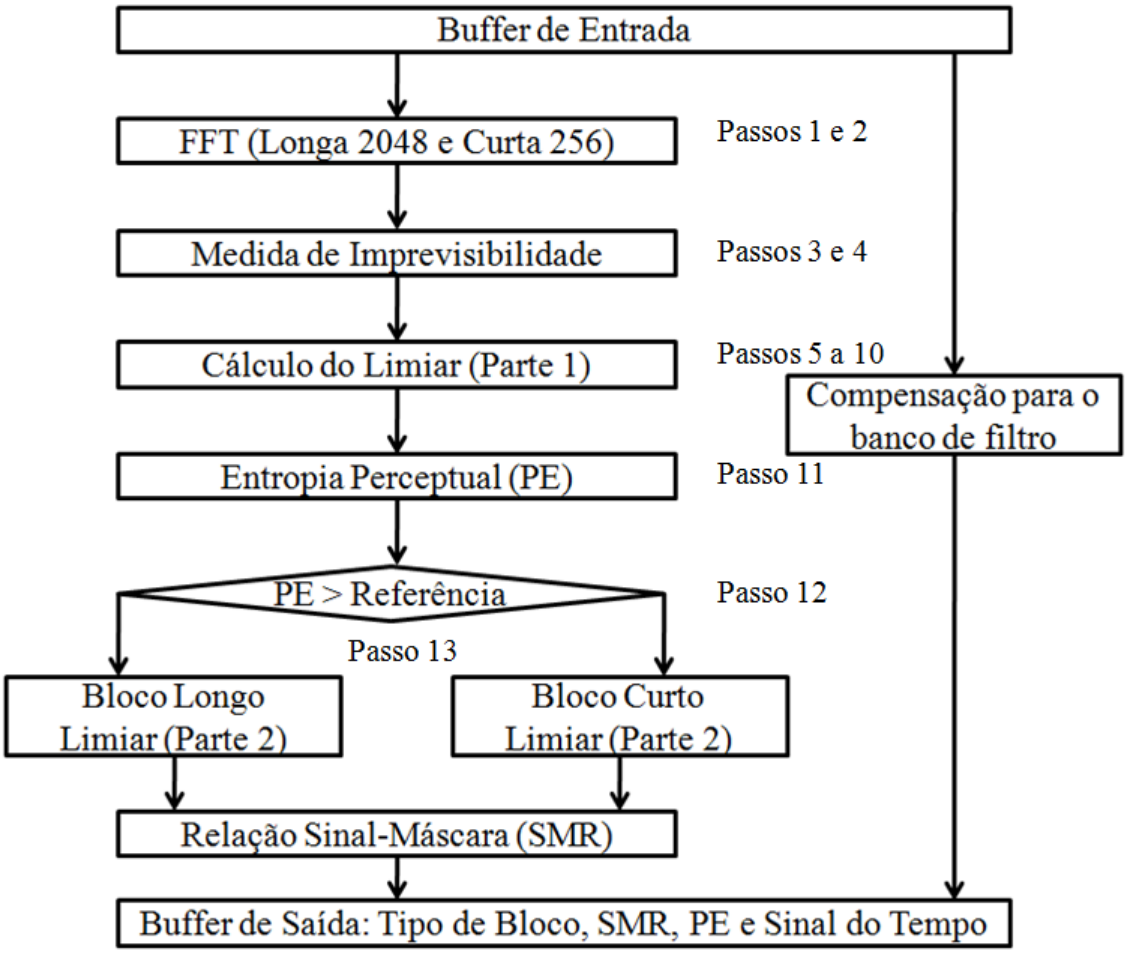


Figura 2.2 - Diagrama de blocos do PAM [2].

Nos passos 1 e 2 o PAM normaliza as amostras no domínio do tempo como entradas para depois multiplicá-las pela janela de Hann e transformá-las para o domínio da frequência através da Transformada Rápida de Fourier (FFT). O resultado final desses cálculos resulta no espectro complexo de:

$$X(k) = \sum_{n=0}^{N-1} w(n)x(n)e^{-j(2\pi kn/N)} \tag{2.1}$$

Depois de calculado este espectro, é necessário representar $X(k)$ em coordenadas polares, resultando em

$$X(k) = r(k)e^{j\emptyset(k)} \tag{2.2}$$

onde $r(k)$ representa a magnitude e $\emptyset(k)$ a fase do componente espectral.

De posse do resultado da FFT, nos passos 3 e 4 sãofeitos os cálculosda energia total da banda, que pode ser obtida por meio do somatório da energia dos componentes espectrais de cada banda e o cálculo da medida de imprevisibilidade através dos espectros da parte imaginária da FFT.

Em seguida, no passo 5, são utilizados os valores da energia particionada e da medida de imprevisibilidade calculados anteriormente, para serem convoluídos com a função de espalhamento. Usando essa função, é possível determinar os limiares de ruído de mascaramento da banda. A função de espalhamento é calculada várias vezes dentro de um mesmo quadro, pois tanto a energia particionada quanto a imprevisibilidade são convoluídas com a função de espalhamento a fim de estimar os efeitos em todas as demais bandas.

O passo 6 é responsável por estimar o coeficiente de tonalidade de cada banda que será usado para ajustar o mascaramento do sinal e o passo 7 é responsável pelo cálculo da relação sinal ruído (SNR) que, segundo Leite destaca em [6], a atual versão do PAM utilizado no MPEG-4 considera que para o cenário de ruído mascarando tom, um sinal será mascarado se estiver 6 dB abaixo do sinal mascarador, independente de qual for a banda. Já no caso de um tom mascarando ruído, o limiar a ser usado é de 18 dB. Assim é possível calcular a relação sinal-ruído (SNR) da seguinte forma:

$$SNR(b) = 18t_b(b) + 6\big(1 - t_b(b)\big) \tag{2.3}$$

onde $t_b(b)$ representa o coeficiente de tonalidade da banda b.

Nos passos de 8 à 10 é realizado o cálculo do limiar da energia de mascaramento e utilizar este resultado para calcular a curva de mascaramento. Como mecanismo para diminuir os efeitos de pré-eco, Leite em [6] também relata que o modelo perceptual do AACcompara o limiar de mascaramento atual ao limiar do bloco anterior.

Nos passos 11 e 12 é feito o cálculo da entropia perceptual que representa a quantidade de informação relevante em um sinal de áudio, em *bits* por amostra para atingir-se a codificação transparente [6]. A partir desse resultado é determinado o tipo de bloco a ser usado pela MDCT. O algoritmo de escolha do tipo de bloco é mostrado por meio da Figura 2.3.

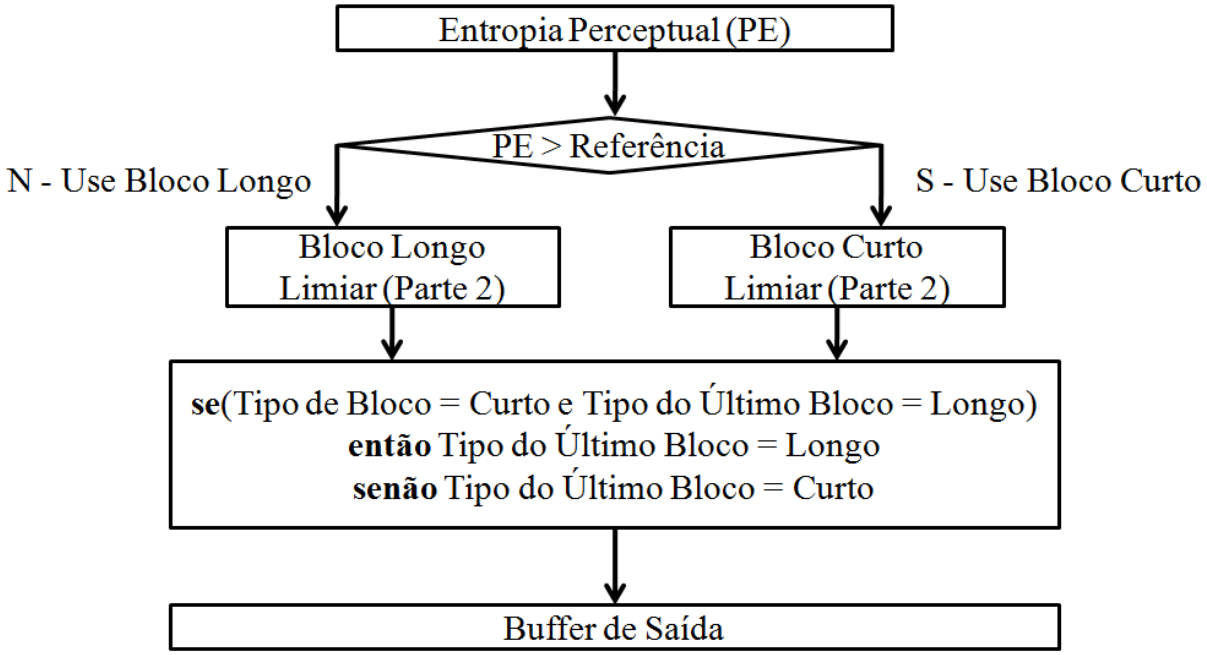


Figura 2.3 – Exemplo de função de mudança de tipo de bloco.

O último passo (13) a ser executado no PAM é o cálculo da relação sinal-máscara (SMR) que, de forma análoga a SNR já calculada, define a relação entre a energia do sinal e o limiar de mascaramento.

2.2.2Controle de Ganho

Esta ferramenta é composta por um banco de filtros *Polyphase Quadrature Filter*(PQF)de quatro bandas uniformes, e de detectores e modificadores de ganhos. Permite a decomposição e normalização do sinal de áudio, o que leva a uma representação escalável em frequência em quatro níveis, o que vai facilitar as operações seguintes.

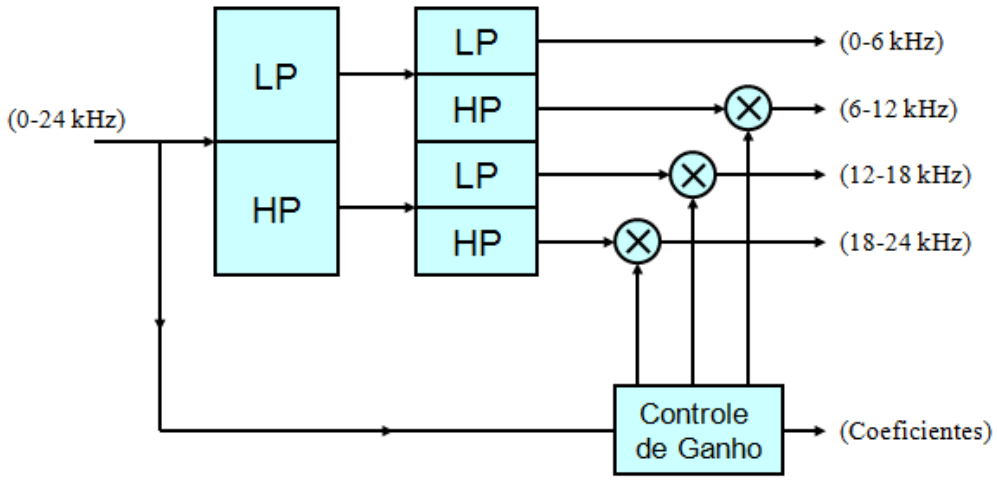


Figura 2.4- Diagrama de blocos do controle de ganho[7].

O uso do controle de ganho, tanto na codificação quanto na decodificação, é designado para reduzir os efeitos do pré-eco para sinais de entrada transientes, atenuando ou amplificando a saída de cada banda PQF.

2.2.3 Banco de Filtros e *Block Switching*

Um componente fundamental no processamento de codificação de áudio é a conversão dos sinais no domínio do tempo em uma representação tempo-frequência. Essa conversão é feita pela MDCT.

Tal transformada toma o bloco apropriado de amostras no tempo, modula-os por uma função de janelamento adequada e executa a DCT. Cada bloco de amostras de entrada é sobreposto 50% com o bloco imediatamente anteriore com o bloco seguinte.O tamanho do bloco de entrada transformado *N*, pode ser configurado para 2048 ou 256 amostras.

2.2.3.1 Janelamento e *Block Switching*

A decisão do formato da janela é feita pelo codificador em uma análise quadro a quadro.A janela selecionada é aplicada apenas na segunda metade da função de janelamento. A primeira metade é obrigada a utilizar o formato da janela do quadro anterior.Por exemplo, se o tamanho do bloco for 2048, o valor no domínio do tempo $x'_{i,n}$ vai ser janelado com os últimos 1024 valores do bloco anterior concatenados com os 1024 valores do bloco atual. A equação abaixomostra como é feita esta concatenação [2]:

$$x'_{i,n} = \begin{cases} x_{(i-1)}, \ para\ 0 \leq n \leq 1024 \\ x_{i,n}, \ para\ 1024 \ \leq n \leq 2048 \end{cases} \tag{2.4}$$

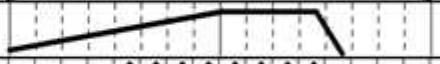
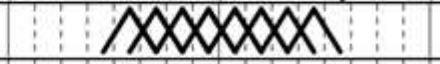
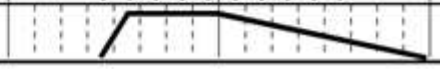
onde *i* é o índice do bloco e *n* é o índice de amostras dentro de um bloco.

Uma vez que o formato da janela é selecionado, dois campos de sintaxe são inicializados com todas as informações necessárias para a criação da janela. São eles:

-**window_sequence**: campo formado por dois *bits* que indicam o tamanho do bloco utilizado. Os possíveis valores são apresentados na Tabela 2.2.

-**window_shape**: campo formado por um *bit* que indica qual função de janelamento é selecionada.

Tabela 2.2- Sequências de janelas [2].

value	window_sequence	num_ windows	looks like
0	ONLY_LONG_SEQUENCE = LONG_WINDOW	1	
1	LONG_START_SEQUENCE = LONG_START_WINDOW	1	
2	EIGHT_SHORT_SEQUENCE = 8 * SHORT_WINDOW	8	
3	LONG_STOP_SEQUENCE = LONG_STOP_WINDOW	1	

Conforme descrito em [2], dependendo dos valores assumidos em **window_sequence** e **window_shape** diferentes funções de janelamento serão utilizadas.

Para **window_shape** = 1, os coeficientes da janela são dados pela janela derivada de Kaiser-Bessel (KBD) como segue:

$$W_{KBD\_LEFT,N}(n) = \sqrt{\frac{\sum_{p=0}^{n}[W\prime(p,\propto)]}{\sum_{p=0}^{N/2}[W\prime(p,\propto)]}}, \ para\ 0 \leq n \leq \frac{N}{2} \tag{2.5}$$

$$W_{KBD\_RIGHT,N}(n) = \sqrt{\frac{\sum_{p=0}^{N-n-1}[W\prime(p,\propto)]}{\sum_{p=0}^{N/2}[W\prime(p,\propto)]}}, \ para\ \frac{N}{2} \leq n \leq N \tag{2.6}$$

onde, *W'* é definido como:

$$W'(n,\propto) = \frac{I_0\left[\pi\propto\sqrt{1.0-\left(\frac{n-N/4}{N/4}\right)^2}\right]}{I_0[\pi\alpha]}, \ para\ 0 \leq n \leq \frac{N}{2} \tag{2.7}$$

e $I_0[x]$ é a função de Bessel modificada de ordem zero do primeiro tipo:

$$I_0[x] = \sum_{k=0}^{\infty} \left[\frac{\left(\frac{x}{2}\right)^k}{k!} \right]^2, \quad (2.8)$$

e $\alpha = kernel\ window\ alpha\ factor, \alpha = \begin{cases} 4\ para\ N = 2048 \\ 6\ para\ N = 256 \end{cases}$

Caso contrário, para **window_shape** = 0, uma janela seno é empregada da seguinte forma:

$$W_{SIN\_LEFT,N}(n) = \sin\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right) \quad para\ 0 \le n \le \frac{N}{2} \quad (2.9)$$

$$W_{SIN\_RIGHT,N}(n) = \sin\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right) \quad para\ \frac{N}{2} \le n \le N \quad (2.10)$$

O tamanho da janela *N* pode ser 2048 ou 256 para KBD e janela seno. Para todos os tipos de **window_sequences,**o**window_shape** da primeira metade da janela vai ser determinado pelo tipo de janela utilizado no bloco anterior:

$$W_{LEFT,N}(n) = \begin{cases} W_{KBD_{LEFT},N}(n),\ se\ window\_shape\_previous\_block == 1 \\ W_{SIN_{LEFT},N}(n),\ se\ window\_shape\_previous\_block == 1 \end{cases} \quad (2.11)$$

Um típico cenário de janelas composto por diferentes estados é mostrado na Figura 2.5. Na figura podem ser vistos os seguintes estados: **ONLY_LONG_SEQUENCE**(a)**, LONG_START_SEQUENCE**(b)**, EIGHT_SHORT_SEQUENCE**(c)**, LONG_STOP_SEQUENCE**(d)e**ONLY_LONG_SEQUENCE**(e)**.**

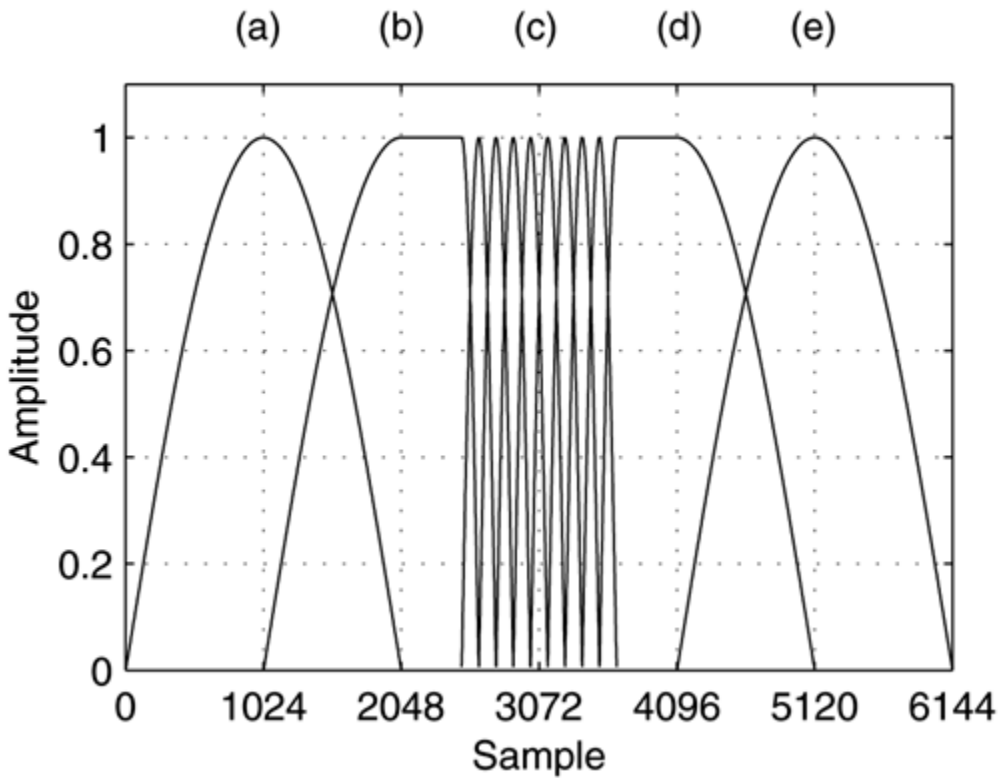


Figura 2.5 - Sequência típica de mudança de janelas[8].

2.2.3.2 *Modified Discrete Cosine Transform*

A MDCT é uma transformada baseada na transformada do cossenodiscreta, com a propriedade de janelamento e adicionalmente a sobreposição. A transformada é definida para ser executada em blocos consecutivos de um conjunto de amostras, onde os blocos subsequentes são sobrepostos de modo que a última metade de um bloco coincida com a primeira metade do próximo bloco. Esse processo permite a reconstrução do sinal sem descontinuidade de blocos, por meio de cancelamento e superposição (a*liasing*) no domínio do tempo, técnica conhecida como *Time-DomainAliasing Cancellation*(TDAC)" [10].Essa sobreposição somada com as qualidades de compactação de energia presentes na DCT faz a MDCT especialmente atrativa para aplicação de compressão de sinais.

A MDCT é uma transformada que tem uma grande utilização especificamente na codificação com perdas devido à concentração dos coeficientes mais significativos da transformada no início do vetor de dados. Essa é uma função linear $F: R^{2N} \rightarrow R^{N}$ eos coeficientesespectrais, $X_{i,n}$, são definidos da seguinte forma:

$$X_{i,k} = 2\sum_{n=0}^{N-1} z_{i,n} \cos\left(\frac{2\pi}{N}(n+n_0)\left(k+\frac{1}{2}\right)\right),\ para\ 0 \le k \le \frac{N}{2} \quad (2.12)$$

onde:

$z_{i,n}$ = sequência de entrada janelada;

n = índice da amostra;

k = índice do coeficiente espectral;

i = índice do bloco;

$$N = \begin{cases} 2048,\ se\boldsymbol{ONLY\_LONG\_SEQUENCE}(0x0) \\ 2048,\ se\boldsymbol{LONG\_START\_SEQUENCE}(0x1) \\ 256,\ se\boldsymbol{EIGHT\_SHORT\_SEQUENCE}(0x2) \\ 2048,\ se\boldsymbol{LONG\_STOP\_SEQUENCE}(0x3) \end{cases} \quad (2.13)$$

$$n_0 = \left(\frac{N}{2}+1\right)/2 \quad (2.14)$$

2.2.4*Temporal Noise Shaping*

A ferramenta *Temporal Noise Shaping* (TNS)representa um novo conceito na codificação de áudio perceptual. O principal fato que motivou a criação dessa ferramenta,segundo [9],é a complexidade da codificação de sinais de áudio e fala devido ao descasamento temporal entre os limiares de mascaramento e o ruído de quantização. Isso acontece porque o ruído de quantização é distribuído uniformemente dentro de cada janela do banco de filtros da transformada, enquanto o limiar real de mascaramento varia consideravelmente nesse período de tempo. Sua principal função é suavizar o ruído dequantização ao longo do tempo.

Os conceitos envolvidos na técnica TNS podem ser caracterizados pelos seguintes aspectos principais [8]:

- Dualidade entre tempo e frequência:

A utilização dessa dualidade é feita para ampliar ainda mais as técnicas de codificação preditiva conhecidas.Jayant em[10]reforça o conceito de que os sinais com um espectro não plano podem ser codificados eficientemente tanto por ação direta da codificação dos valores espectrais ou pela aplicação de métodos de codificação preditiva para o sinal de tempo.

A codificação eficiente de sinais transitórios pode serconseguidapor meio da

codificação de valores no domínio do tempo ou pela aplicação de métodos de codificação preditiva para os dados espectrais.Na verdade pode ser mostrado que devido a essa dualidade entre tempo e frequência, a quantidade de ganho de predição (redução da energia residual) alcançado é determinada pelo não nivelamento do envelope temporal do sinal.

- Modelagemdo Ruído através da Codificação Preditiva:

A Figura 2.6 mostra o funcionamento do TNS. Primeiramente é aplicado um filtro no espectro original, seguido de um filtro de pré-ênfase e, em seguida, há uma quantização neste sinal processado. Na saída, os coeficientes quantizados são transmitidos no fluxo de *bits* e são utilizados na decodificação de forma inversa.

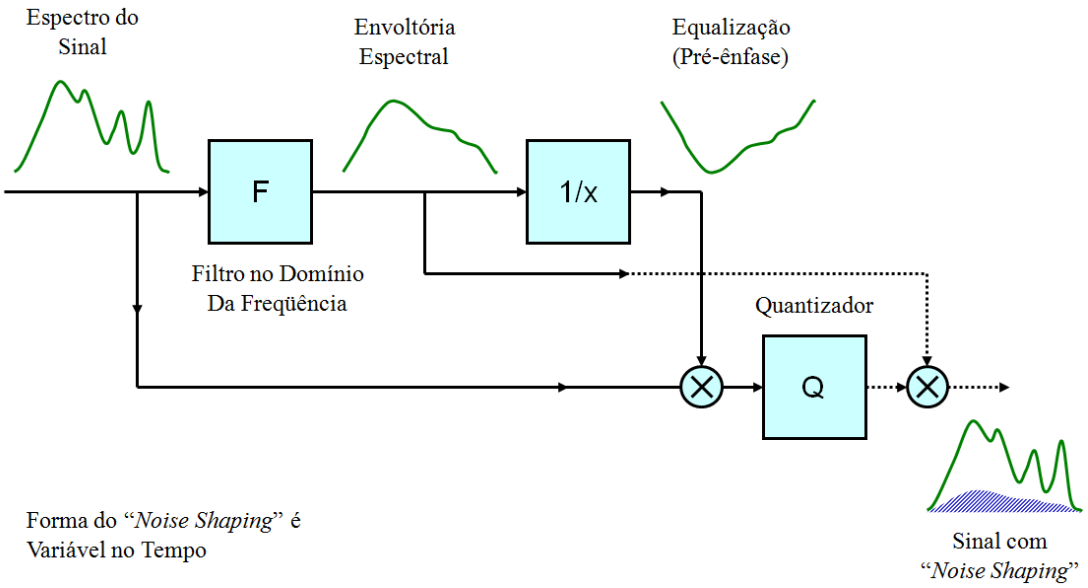


Figura 2.6 - *Temporal Noise Shaping*[7].

2.2.5*Intensity/Coupling*

Semelhante ao que ocorre no *Parametric Stereo,* o MPEG faz uso de outros mecanismos de codificação multicanal de sinais de áudio com baixas taxas de *bits*, conhecidos por codificação conjunta. O processo de codificação conjunta é efetuado no padrão MPEG em diferentes momentos e cada um irá atuar em diferentes regiões de frequência, podendo assim ter seus resultados somados e combinados.

2.2.5.1 *Intensity Stereo Coding*

A primeira ferramenta de codificação conjunta aplicada é o *Intensity Stereo Coding*cuja codificação é realizada entre dois canais de áudio. Dessa forma é gerado um único canal de saída composto de um conjunto de coeficientes espectrais. O fato de a audição humana variar em função da frequênciajustifica a utilização dessa ferramenta.Assim, ao explorar essa "limitação", o *Intensity Stereo Coding* pode reduzir a taxa de um fluxo de dados com nenhuma ou quase nenhuma mudança perceptível na qualidade aparente.

Os coeficientes espectrais compartilhados são transmitidos na parte do canal esquerdo e no canal direito são carregados os valores de intensidade, indicando o dimensionamento do canal direito em comparação ao canal esquerdo.

O *Intensity Stereo Coding* pode ser dito como uma técnica de PS (*Parametric Stereo*), no entanto, permite apenas controlar a localização lateral do som usando um parâmetro que atua selecionando a quantidade de sinal de áudio a ser enviado para cada canal, não havendo necessidade de recriar o ambiente estéreo presente no sinal original.

2.2.5.2 *Coupling Channels*

Outro mecanismo um pouco mais complexo de codificação conjunta é o *Coupling Channels*. A complexidade deste mecanismo se justifica por ele não trabalhar somente com pares de canais, podendo ser aplicado também em ambientes multicanais. Essa ferramenta tem duas funcionalidades básicas,segundo [2].

A primeira delas é a possibilidade de implementação do *intensity stereo coding*por meio do *coupling channels*. Essa possibilidade pode ocorrer quando os espectros dos canais podem ser compartilhados por meio dos limites do canal.A segunda funcionalidade é a realização dinâmica de um *downmix*entre vários canais. Assim haverá um número menor de dados a serem transmitidos.

2.2.6*Prediction*

O bloco*Prediction,* mostrado naFigura 2.1, é mais um bloco presente no MPEG-4 no qual técnicas de predição são utilizadas como forma de explorar a redundância entre quadros sucessivos em um sinal.

Essa técnica já fazia parte do padrão MPEG-2 AAC e tem como princípio a ideia de que com base nos coeficientes espectrais quantizados do quadro anterior é possível obter uma estimativa do valor atual por meio de um preditor.Pereira diz em [8]que dessa forma será alcançada uma economia na taxa de *bits* pela quantização e codificação dos resíduos de predição ao invés dos coeficientes espectrais reais. Esse procedimento de remoção de redundância é realizado pelo emprego de umapredição retrógrada de segunda ordem baseada nos coeficientes espectrais dos dois últimos quadros.

A ferramenta *Prediction* implementa a predição ao longo do tempo, o que vai aumentar de forma efetiva o alcance temporal do codificador. Já a ferramenta TNS implementa a predição ao longo da frequência, o que vai proporcionar também de forma efetiva uma maior resolução temporal do codificador. Dessa forma, pode-se dizer que essas ferramentas são duais e se completam em termos de funcionalidade.A utilização desta ferramenta está somente disponível no perfil *Main Audio Profile*, pois ela representa um aumento da complexidade muito alto.

2.2.7*Perceptual Noise Substitution*

No MPEG-4 foi feita a introdução de mais uma nova ferramenta para melhoria do padrão, trata-se do *Perceptual Noise Substitution* (PNS). Esse é um recurso que visa uma otimização da eficiência em baixas taxas de *bits*.Segundo [11],a ferramenta PNS é um exemplo de um método de codificação que não visa a forma de onda. Baseia-se no fato de que um ruído soa como os outros, ou seja, que a estrutura atual de um ruído não é tão importante para a percepção subjetiva de tal sinal.

Ele é projetado para representar o ruído como componentes do sinal de entrada com uma representação paramétrica muito compacta e dessa forma aumenta ainda mais a eficiência da compressão do codificador AAC para sinais de entrada adequados.O *bitstream* teria apenas um sinal de que naquela região haveria um ruído e daria algumas informações adicionais sobre a energia total presente.

No processo de decodificação, para substituir a informação de que naquela região espectral haveria um ruído é gerado um ruído aleatório de acordo com o nível de potência do sinal.

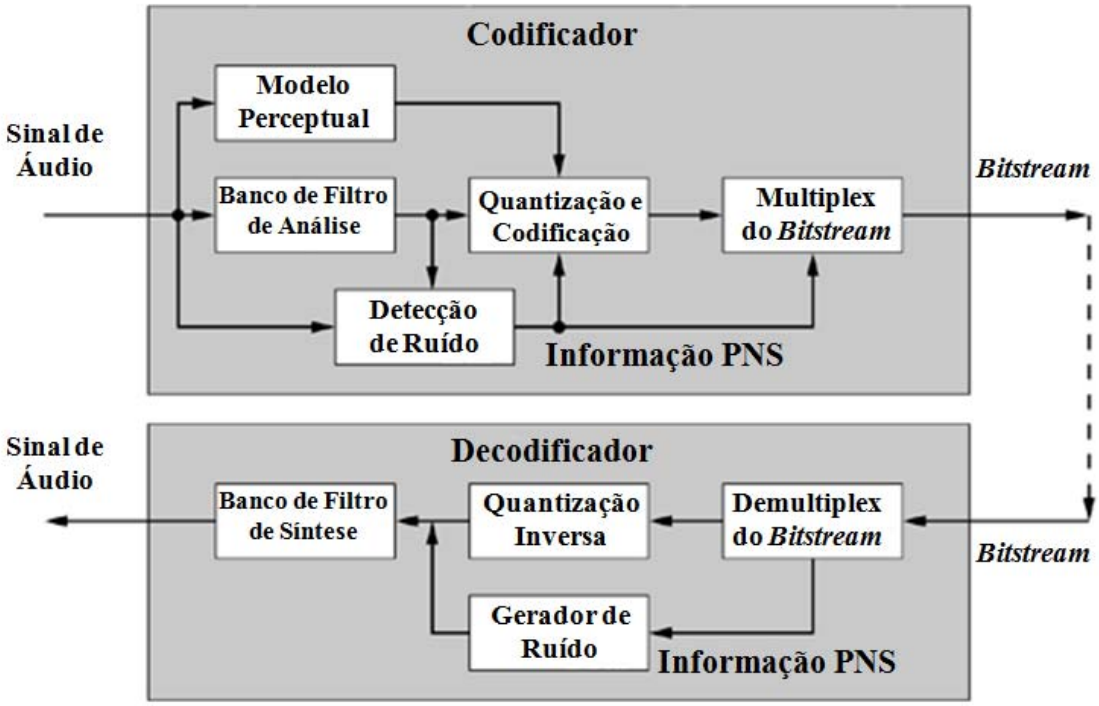


Figura 2.7 - Princípio de codificação com o PNS [8].

A Figura 2.7representa a localização do PNS dentro do modelo AAC. Nota-se que a inserção dessa nova ferramenta foi efetuada pensando em utilizar a arquitetura já existente do codificador AAC, inclusive com a reutilização de alguns mecanismos. Dentre as formas de reutilização de mecanismos empregadas, segundo[5],podem ser citadasque:

- No processo de decodificação o uso da ferramenta PNS é sinalizado pelo uso do pseudo *codebook*`NOISE_HCB=13.`

− Se o mesmo fator de normalização de banda e grupo for codificado pelo PNS em ambos canais de um par, nenhuma decodificação conjunta é realizada para esse fator de normalização da banda e grupo.

- Os coeficientes do "pseudo-ruído" gerado pela ferramenta PNS são injetados no espectro de saída antes da etapa de processamento TNS.

2.2.8*Mid/Side*

A técnica de codificação de sinais estéreos *Mid-Side* (M/S) é a segunda ferramenta utilizada no padrão MPEG-4 como forma de realizar a codificação conjunta de canais de áudio. Assim como o *Intensity/Coupling* tem a função de unir dois canais com a finalidade de reduzir a

complexidade.Trata-se de uma ferramenta que realiza a codificação conjunta em pares de canais. Para isso, é aplicada uma matriz para os sinais do canal direito e esquerdo, realizando a soma e a diferença desses dois sinais originais. Dessa forma, ao invés de transmitir os canais direito e esquerdo separadamente, só será codificado os sinais depois da soma e diferença entre os dois canais, por exemplo.

O fato do M/S ser uma técnica que só é utilizada em pares de canais não impede que seja utilizada pelo padrão MPEG-4 em ambientes multicanais. Para isso a técnica será utilizada dentro de cada par de canais que estiverem dispostos simetricamente no eixo esquerdo ou direito do ouvinte.Também é uma ferramenta semelhante ao PS, mas pelo fato de basear-se na redundância perceptual entre canais, o PS leva muito mais informações do que a codificação M/S.

2.2.9Quantização e Codificação

O processo de codificação e quantização é um dos pontos fortes do padrão AAC. Por meio dele é obtida uma alta taxa de redução de *bits*, pois utiliza um método de quantização adaptativa. Os componentes mais relevantes nesse processo de quantização são na verdade, as funções de quantização e a formação de ruído que é alcançada por meio dos fatores de escala. O esquema de quantização baseado na lei da potência é regido pela equação

$$ix(i) = sign\big(x(i)\big) * nint\left[\left(\frac{|x(i)|}{\sqrt[4]{2}^{scale\_factor}}\right)^{0.75} - \alpha\right] \qquad (2.15)$$

onde $x(i)$ e $ix(i)$ denotam os valores não quantizados e quantizados, $sign()$ retorna o sinal de seu argumento, $nint()$ retorna o inteiro mais próximo do valor real do argumento, α é uma pequena constante e $scale\_factor$ refere-se a um parâmetro de resolução de quantização.

Esse modelo de quantização já vinha sendo aplicado em padrões antecessores de forma semelhante.

2.2.9.1 Fatores de Escala

Mesmo já existindo uma modelagem de ruídos no quantizador não linear, normalmente apenas esse quantizador não é suficiente para alcançar uma qualidade de áudio tida como transparente. Para melhorar a qualidade subjetiva do sinal codificado, o ruído é novamente

modelado por meio dos fatores de escala.Os fatores de escala são utilizados para a modelagem dos ruídos de quantização no domínio da frequência. Para isso, o espectro é dividido em vários grupos de coeficientes espectrais chamados banda para fatores de escala (*Scalefactors Bands*) que compartilham um fator de escala.

Com base em [5], pode-se dizer que o fator de escala é um multiplicador que representa um valor de ganho que é usado para mudar a amplitude de todos os coeficientes espectrais na banda. Esse mecanismo é usado para mudar a alocação do ruído de quantização no domínio espectral gerado pelo quantizador não-uniforme.

Cada banda tem um fator de escala que representa um ganho a ser aplicado em todos os coeficientes espectrais daquela banda. Essa estrutura é construída de forma a simular as bandas críticas do sistema de audição humano. Por isso, a quantidade de fatores presentes em uma banda vai variar de acordo com o comprimento da transformada e da frequência de amostragem.

Em alguns casos, será possível que apenas um conjunto de fatoresde escala seja transmitido para várias janelas agrupadas. Isso vai ocorrer se houver uma sequência que contenha apenas **SHORT_WINDOWS.**

A Figura 2.8mostra um caso de agrupamento de **SHORT_WINDOWS.**

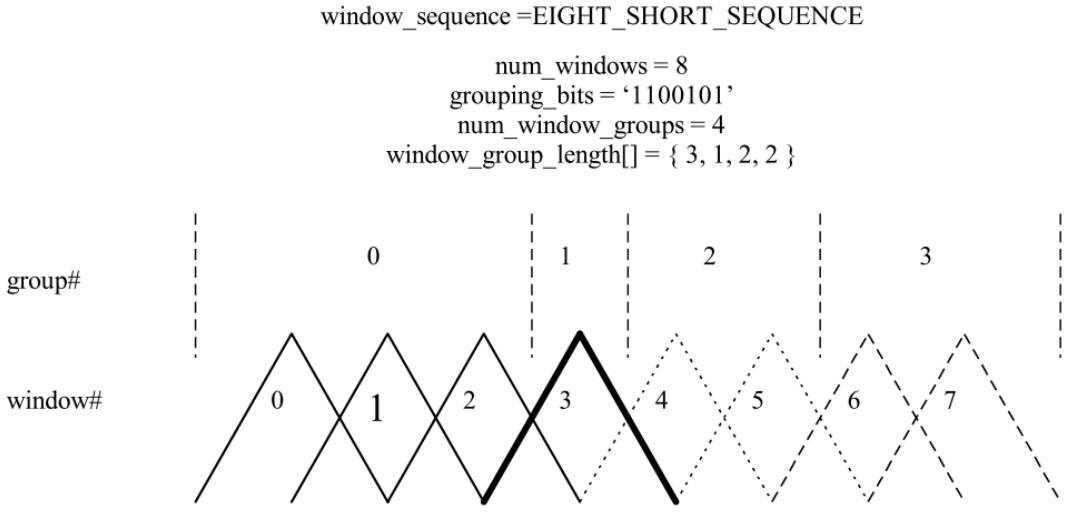


Figura 2.8 - Exemplo de agrupamento de janelas curtas [5].

Após tal agrupamento das janelas, a Figura 2.9 mostra a economia de dados a serem enviados através da comparação de um caso onde somente existam janelas de tamanho grande

(**ONLY_LONG_SEQUENCE**) com o exemplo de oito janelas pequenas (**EIGHT_SHORT_SEQUENCE**).

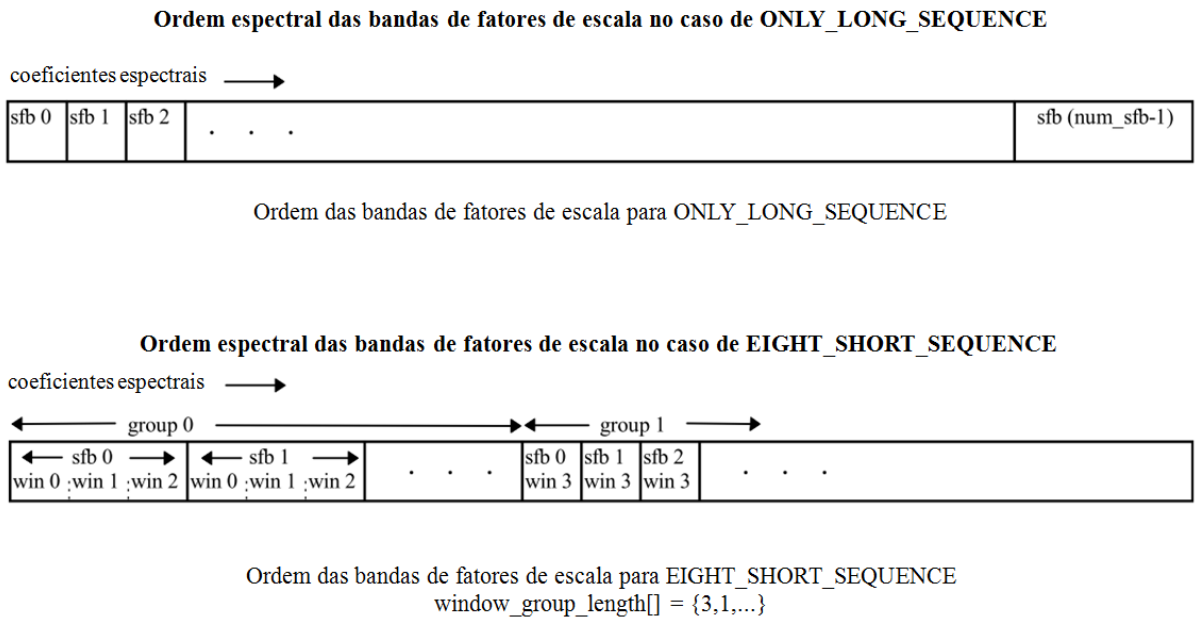


Figura 2.9 - Comparação de casos com diferentes tipos de janelas [5].

2.2.9.2Codificação Sem Ruído

Segundo [12],a codificação sem ruído codifica o conjunto de 1024 coeficientes espectrais quantizados da forma mais eficiente possível explorando a redundância estatística sem qualquer redução de precisão. Esse processo utiliza um método simples de redução de faixa dinâmica, fazendo a codificação de até quatro coeficientes e substituindo-os por valores de +/-1 na matriz de coeficientes quantizados. Já, segundo Pereira em [8],as magnitudes desses coeficientes são codificadas separadamente, juntamente com os índices de frequência correspondente.No entanto, o mecanismo geral de codificação na codificação sem ruído é baseado nas técnicas de seccionamento e codificação de Huffman definidas em[13].

2.2.9.3 Codificação de Huffman

A codificação de Huffman é realizada com base na probabilidade de ocorrência de um símbolo dentro de um conjunto de dados. Por meio desta probabilidade é que se determina o tamanho de cada código, mesmo sendo um número que pode variar este tamanho ainda deve ser inteiro.

No padrão do AAC, um código de Huffman estendido é usado para representar *n-tuples* de coeficientes quantizados com os *codewords* retirado de um dos 11 *codebooks*. Os coeficientes espectrais dentro das *n-tuples* são ordenados e o tamanho de cada um é de dois ou quatro coeficientes segundo [12].

A Tabela 2.3mostra o valor máximo dos coeficientes quantizados que podem ser representados por cada *codebook* de Huffman e o número de coeficientes em cada *n-tuple* para cada *codebook*.

Tabela 2.3 - Códigos de Huffman do AAC

Índice do *Codebook*	Tamanho do *Tuple*	Maior Valor Absoluto	Sinalizado ou Não Sinalizado
00	-	00	
01	4	01	Sinalizado
02	4	01	Sinalizado
03	4	02	Não Sinalizado
04	4	02	Não Sinalizado
05	2	04	Sinalizado
06	2	04	Sinalizado
07	2	07	Não Sinalizado
08	2	07	Não Sinalizado
09	2	12	Não Sinalizado
10	2	12	Não Sinalizado
11	2	16	Não Sinalizado

Dentre todos os *codebooks* mostrados na Tabela 2.3, dois têm características especiais. O *codebook*0 indica que todos os coeficientes dentro de uma seção são zeros. O *codebook*11 pode representar coeficientes que tem um valor absoluto maior ou igual ao valor máximo de 16.

2.2.9.4 Seccionamento

O seccionamento é um processo dinâmico para que o codificador se adapteàs condições de variação que acontece com os coeficientes dependendo da frequência e do tipo do sinal. É feito usando um algoritmo ganancioso (*greedy algorithm*) que começa com o número máximo de seções possíveis, cada qual usando o *codebook* de Huffman com o menor índice

possível. As seções são então misturadas. Se essas seções não usarem o mesmo *codebook* então o*codebook* com um índice igual em pelo menos dois casos pode ser usado. Esse é o único corte de informações realizado no processo de codificação sem ruído.

Além do esquema de quantização e codificação apresentado, o AAC tem outras ferramentas que também podem ser utilizadas com a finalidade de obter ganhos cada vez maiores. Essas ferramentas não serão citadas por não estarem presentes no perfil HE-AACv2.

Capítulo 3

A arquitetura do *Digital Signal Processor*

Nos capítulos anteriores foi feita uma breve introdução ao padrão de áudio da suíte MPEG-4, discorrendo sobre sua evolução e sobre suas principais ferramentas. Este capítulo faz uma introdução ao processador digital de sinais (DSP) utilizado, focando em uma apresentação de sua arquitetura e de seu modo de funcionamento.

Para esta implementação foi utilizado um DSP da família ADSP-BF5xx Blackfinda empresa Analog Devices (AD).

3.1 DESCRIÇÃO DO PROCESSADOR BLACKFIN

Os processadores ADSP-BF5XX Blackfinsão uma família de processadores baseados na arquitetura*Micro Signal Architecture*(MSA) desenvolvida pela Analog Devicesem parceria com a Intel, que reúne as características de um micro controlador e de um processador DSP.

Segundo Gan [14]a grande vantagem do núcleo MSA é a integração de características que combinam processamento multimídia, comunicação e interface com o usuário em uma simples plataforma. Esses DSP são processadores de ponto-fixo que operam com 16/32 *bits* e tem um baixo consumo de energia. Seu núcleo combina um duplo mecanismo de processamento de sinais MAC (multiplicador e acumulador), um conjunto de instruções para microprocessador *Reduced Instruction Set Computing*(RISC), e recursos de multimídia em uma arquitetura com um único conjunto de instruções.

A Figura 3.1 apresenta o diagrama de blocos do DSP ADSP-BF527. Nela são vistos vários periféricos que são conectados ao núcleo através de vários barramentos. Dentre esses periféricos do sistema, pode ser citado o *Parallel Peripheral Interface*(PPI), *Serial Peripheral Interface*(SPI), *Serial Ports* (SPORTs), timers, *Universal Asynchronous Receiver Transmitter* (UART), além de outras portas de comunicação bidirecional.

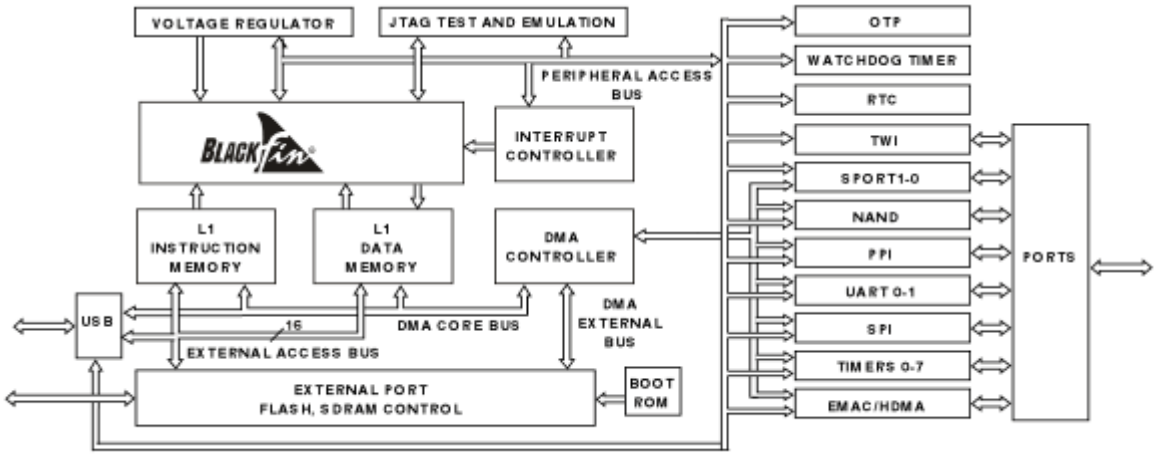


Figura 3.1 - Diagrama de blocos do Blackfin ADSP-BF527

Além desses periféricos, o DSP também possui um controlador de *Direct Memory Access*(DMA) que efetivamente transfere dados entre dispositivos/memórias externos e as memórias internas sem intervenção do núcleo do processador.

3.1.1 Núcleo do Processador Blackfin

A arquitetura do núcleo do processador pode ser vista através da Figura 3.2.O núcleo dos processadores Blackfin é composto basicamente por três unidades principais:

- Unidade aritmética de endereços;
- Unidade aritmética de dados;
- Unidade de controle.

O núcleo contém dois multiplicadores de 16 *bits*, dois acumuladores de 40 *bits*, duas Unidades Lógicas Aritméticas de 40 *bits* (ULA), quatro ULAs de vídeo de 8 *bits* e um *shifter* de 40 *bits*. Também possui oito registradores de 32 *bits* de dados, que podem ser usados tanto com 32 ou 16 *bits* e possui dois *Data Address Generators*(DAG) que auxiliam o DSP na manutenção do transporte de dados entre memória e os registradores do núcleo. Toda esta arquitetura permite que sejam realizadas mais de uma operação de multiplicação e adição em um único ciclo de *clock*.

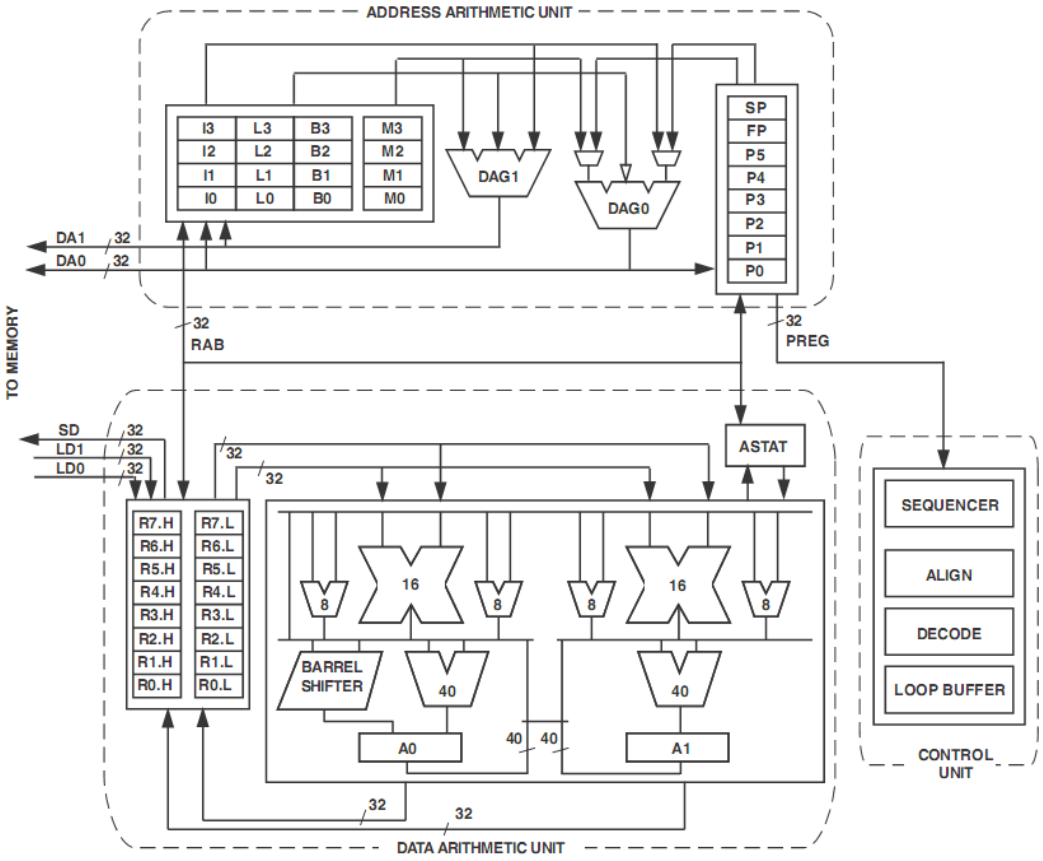


Figura 3.2 - Núcleo do processador Blackfin [15].

3.2 ARQUITETURA DE MEMÓRIA

O processador Blackfin possui um sistema de memória eficaz composto de memória interna e externa. Para o processador, não existe distinção entre essas memórias. Ele apenas vê a memória como um simples espaço de endereçamento de 4 Gb usando endereçamento de 32 *bits*. A estrutura hierárquica da memória no Blackfin é vista na Figura 3.3.

Figura 3.3 - Modelo hierárquico de memória do Blackfin [14].

As porções de memória são arranjadas em uma estrutura hierárquica onde se busca uma boa relação de balanço entre memórias internas pequenas, caras e de baixa latência com memórias grandes, porém com menor custo e mais lentas. A Figura 3.4apresenta a distribuição de endereçamento padrão de memória interna e externa do ADSP-BF527.

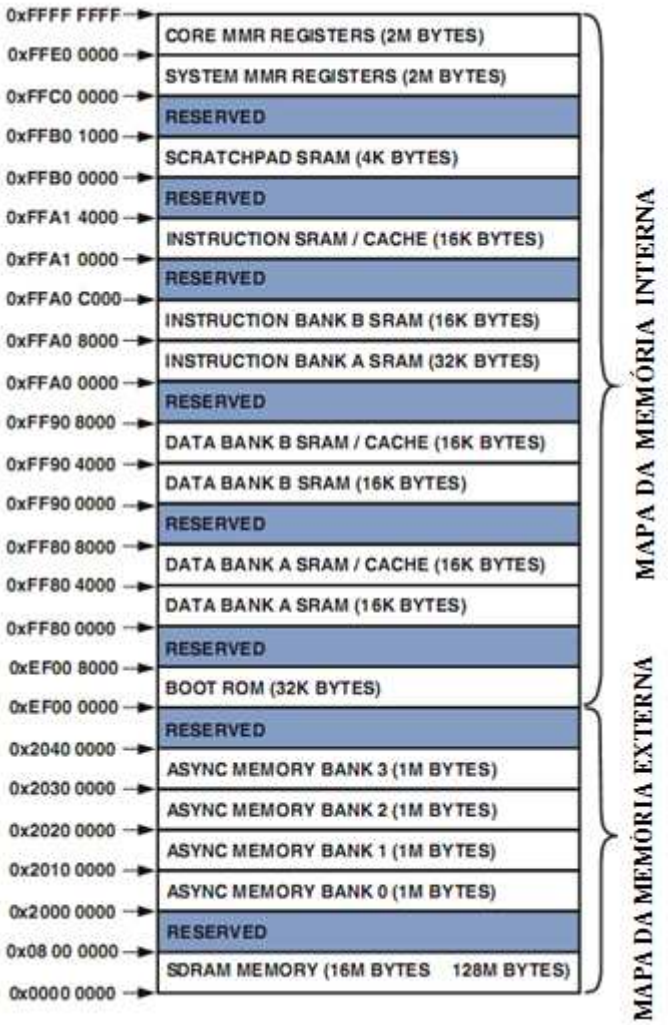


Figura 3.4 - Mapa de memória interna e externa do Blackfin ADSP-BF527 [15].

3.2.1 Memória Interna

O banco de memória interna (L1) é composto de três blocos de memória *on-chip*.O primeiro banco é a memória de instruções L1, composta de 64 kb de memória *Static Random Access Memory* (SRAM), dos quais 16 kb podem ser configurados como memória *cache*.O segundo banco é a memória de dados L1, composta de até dois bancos de 32 kb cada um. Já o terceiro banco é uma memória*scratchpad*SRAM de 4 kb. Este banco não pode ser configurado

como *cache* ou como acesso DMA.

3.2.2 Memória Externa

A memória externa do Blackfin (L2) é uma memória *off-chip*. Para ser acessada é utilizado o barramento *External Bus Interface Unit*(EBIU). Esta interface de 16 *bits* fornece conexão para um banco de memória *Synchronous Dynamic Random Access Memory*(SDRAM) e também para outros quatro bancos de memória assíncronos.

O tamanho do banco SDRAM pode ser modificado, variando de 16 Mbytes até 128 Mbytes e sempre corresponde aos primeiros endereços no banco de memória (0x0000 0000 até 0x0800 0000).

Os outros quatro bancos de memória assíncrona podem ser tanto memória *flash,* assim como memória *Read Only Memory* (ROM), *Erasable Programmable Read Only Memory*(EPROM), *Static Random Access Memory* (SRAM) e dispositivos de saída mapeados em memória. Cada um desses bancos tem o tamanho de 1 Mbyte e sempre serão representados nos endereços subsequentes aos bancos SDRAM (0x2000 0000 até 0x2040 0000).

3.3 ACESSO DIRETO À MEMÓRIA

O *Direct Access Memory* (DMA) é uma característica presente nos DSPs da família Blackfin que permite que alguns subsistemas de *hardware* acessem sistemas de memória para leitura e/ou escrita de forma independente e sem a necessidade de interferência do núcleo, o que faz com que esse acesso seja feito de forma rápida e preservando recursos do núcleo para outras finalidades.

As transferências de DMA podem ocorrer entre as memórias internas do processador e os periféricos conectados ao DMA. Além disso, as transferências de DMA podem ser feitas entre qualquer um dos periféricos e os dispositivos externos conectados às interfaces de memória externa, incluindo controladores SDRAM e controladores de memória assíncrona.Dentre os periféricos controlados por DMA podem ser citados*Universal Serial Bus*(USB), UARTs, SPI, PPI e SPORTs. Para cada um destes periféricos existe pelo menos um canal DMA dedicado[16].

Essas transferências são controladas pelo controlador de DMA e suportam transferências tanto unidimensionais (1D) como transferências bidimensionais (2D). O tipo de transferência DMA é inicializado por meio de registradores ou de conjuntos de parâmetros

chamados descritores de blocos.

3.4 O AMBIENTE DE DESENVOLVIMENTO

Os processadores da família Blackfin possuem um ambiente de desenvolvimento de aplicações. Trata-se do VisualDSP++ [17], desenvolvido pela Analog Devicescom o objetivo de fornecer ao desenvolvedor um ambiente de desenvolvimento com várias ferramentas necessárias para programação de sistemas embarcados.

Esse ambiente de desenvolvimento, segundo descrição em [18], fornece um *Integrated Development and Debugger Environment*(IDDE), compilador nas linguagens ANSI C e C++, um montador Assembler, Linker, Kernel, suporte para emulação e simulação.

Um fluxograma típico do desenvolvimento de uma aplicação usando o VisualDSP++ pode ser visto naFigura 3.5.

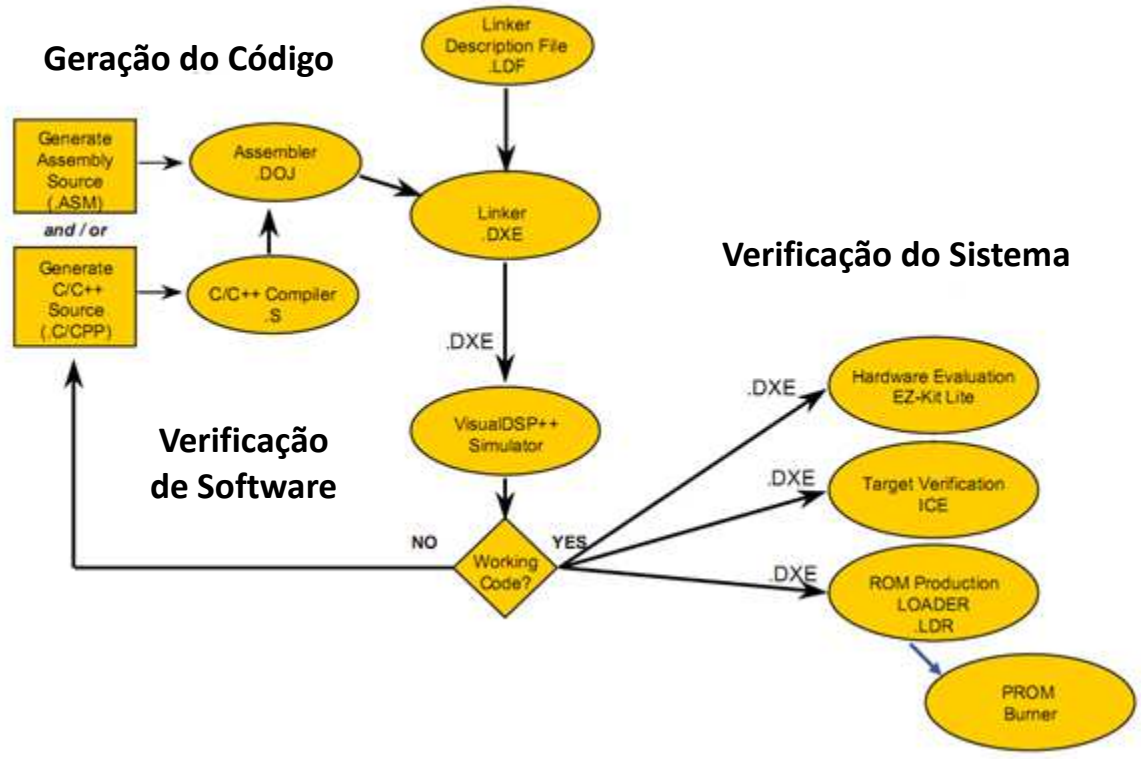


Figura 3.5 - Fluxograma típico do desenvolvimento do *software*[18].

É possível desenvolver aplicações em linguagem C/C++ e/ou *assembly*. Se o código for escrito em linguagem C/C++ o compilador gerará primeiramente um arquivo *assembly* (.s). Em seguida será gerado um arquivo objeto (.doj) pelo montador *assembler* que servirá de entrada

para o *linker*juntar os arquivos objetos para um arquivo executável (.dxe) com o auxílio das informações do arquivo de descrição do *linker* (.ldf), onde constarão várias informações sobre o mapeamento de memória desejado.

Ao final desse processo de geração do código executável, tal arquivo (.dxe) pode ser carregado no DSP para verificar a regularidade do programa. Caso contrário, a depuração do código fonte deve ser realizada e todo o processo será repetido.

3.5 COMUNICAÇÃO ENTRE O DSP E O VISUALDSP++

Após a geração do arquivo executável (.dxe) ele deve ser carregado no DSP. O processo para que isso aconteça pode ser realizado de três diferentes formas pelo VisualDSP++.Uma delas não é uma comunicação propriamente dita, mas trata-se da simulação de uma conexão para a execução de testes quando não há equipamentos de *hardware* disponíveis. O simulador lê o arquivo executável (.dxe) e imita o comportamento de um *chip* DSP para testar e depurar o código antes de embarcá-lo.

O VisualDSP++ simula os dispositivos de memória e de entrada/saída especificados no arquivo .ldf. Além da simulação do programa no DSP é possível simular situações como interrupções aleatórias e transferência de dados.

Outro tipo de comunicação é feita através do EZ-KIT *Lite*, que é um *kit* de desenvolvimento utilizado para avaliar um determinado processador. Esse tipo de comunicação é realizada entre o DSP e o ambiente de desenvolvimentopor meio de uma interface USB, permitindo o controle do comportamento do processador.

A última forma de se comunicar o VisualDSP++ com o DSP é através de um emulador, que é um módulo que controla o processador conectado a um emulador JTAG(*Joint Test Action Group*). Um emulador permite que a aplicação seja baixada e testada dentro do VisualDSP++ de forma rápida fazendo a interação entre a interface JTAG e outras interfaces, como por exemplo, *Peripheral Component Interconnect*(PCI) e USB.

Esse módulo do emulador tem uma interface de comunicação de 14 pinos e usa um superconjunto do padrão IEEE 1149.1 para enviar e receber dados da porta de emulação. O emulador JTAG usa um sinal adicional **EMU~** como um sinalizador de status da emulação para o DSP[19].

Capítulo 4

Técnicas de Otimização

Após a descrição do padrão de áudio HE-AAC v2 e dos DSPs da família Blackfin e suas características o próximo passo é a descrição do processo de implementação do *software* no sistema embarcado. Uma parte importante do porte do código no DSP é a aplicação de técnicas de otimização.

A cada dia é maior o número de aplicações de Processamento Digital de Sinais que exigem um *hardware* dedicado, como um DSP, devido à grande demanda de memória e processamento para sua execução.

Cada DSP tem uma arquitetura de acordo com sua funcionalidade específica e é necessário conhecer e compreender esta arquitetura, além do seu compilador e do código-fonte que está sendo produzido, para que possam ser alcançados melhores resultados.

Para ilustração das diversas técnicas de otimização descritas neste capítulo serão utilizadas algumas funções do decodificador de áudio.Serão discutidas técnicas de otimização em relação à arquitetura do DSP como, por exemplo, uso de memória *cache*, proteção de memória, uso eficiente da memória e gerenciamento dos valores de *clocks*.No que diz respeito ao uso eficiente do compilador serão discutidas técnicas como, por exemplo, uso do tipo de dados mais adequado, funções intrínsecas, funções *inline,* pragmas e uso de técnicas de otimização de *loops*.

4.1 A OTIMIZAÇÃO EM SISTEMAS EMBARCADOS

Otimização é um procedimento que visa maximizar ou minimizar um ou mais índices de desempenho.Dentre esses índices podem ser citados, segundo [20]:

- Taxa de transferência;
- Uso de memória;
- Largura de banda de E/S;
- Dissipação de energia.

A realização da otimização em um ou mais módulos nem sempre representa a melhoria esperada, visto que a otimização de um item pode influenciar na nãomelhora (ou até mesmo a piora) do desempenho de outro item.

Se, por exemplo, a prioridade na otimização for a velocidade, pode haver um aumento do uso de memória, o que dificulta a obtenção de umaaplicação totalmente otimizada.

Em linhas gerais existem duas estratégias principais de otimização para aplicações embarcadas em DSPs que precisam ser consideradas:

- Otimização em relação à arquitetura do DSP;
- Otimização em relação ao compilador do DSP;

As técnicas de otimização em relação ao compilador abordam também o uso de técnicas de otimizações do código-fonte, onde são feitas alterações que visem a utilização do compilador de forma otimizada. Por esse motivo, essas técnicas serão apresentadas de forma conjunta.

4.2 OTIMIZAÇÃO EM RELAÇÃO À ARQUITETURA

Um tipo de otimização que traz excelentes resultados é a otimizaçãodo programa que está sendo desenvolvido de acordo com as configurações da arquitetura do DSP, principalmente no que diz respeito à configuração da arquitetura de memória.

Fazer com que o programa esteja usando as configurações de memória mais adequadas evita acessos, armazenamentos e cálculos desnecessários que poderiam ser evitados. Esse tipo de otimização requer um nível maior de conhecimento da arquitetura em questão mas, caso seja bem realizado, representará grandes melhorias.

Em um ambiente ideal, o programa, os dados e as instruções ficariam todos na memória interna do DSP (L1), mas na implementação em questão e em grande parte das implementações no mundo dos DSPs é necessário o uso de memórias externas ao processador, do tipo L2.

Nos tópicos a seguir, serão feitas considerações mais detalhadas sobre a arquitetura de memória no Blackfin e possíveis alterações com o intuito de obter melhorias no sistema como um todo.

4.2.1 Memória *Cache*

Em se tratando de melhorias em uma aplicação que possui grande número de dados e instruções, o uso de memória *cache* é fundamental. Mesmo com possíveis complicações ou dificuldades de armazenar dados relevantes e mantê-los atualizados em *cache*, esse tipo de memória é bastante utilizada.

Memória *cache* é um componente que armazena dados para que em solicitações futuras o acesso a eles possa ser feito de forma mais rápida. Há alguns conceitos em relação à memória *cache*, independente do seu tipo, que serão fundamentais para sua boa utilização:

- *Cache Hit*: acontece quando o processador se refere a um bloco da memória principalque já está disponível na *cache*.

- *Cache Miss:* acontece quando o processador se refere a um bloco de memória que não está disponível em *cache*. Depois de acontecer isto, o controlador de *cache* move o bloco referenciado para a memória *cache*.

Quando houver a necessidade de se colocar um bloco de entrada em *cache*, este processo pode ser feito de três maneiras. Ou se coloca o bloco em um destino fixo, que é definido com base no endereço do bloco na memória de baixo nível (*Cache*diretamente mapeada), ou se coloca o bloco em um destino completamente aleatório (*Cache*totalmente associativa), ou se arranja a memória em conjuntos onde cada bloco de memória tem um conjunto fixo na *cache* para ser armazenada (*Cache*set associativa).

O controle de*cache miss* é importante para que mais dados possam ser acessados de maneira rápida, o que será de extrema utilidade ao processador. Por isso é importante a escolha da política de preenchimento de *cache* mais adequada possível. Quando não há um endereço fixo para o posicionamento do bloco existem algumas estratégias para o repreenchimento da *cache*. As estratégias são as seguintes [21]:

- *Direct Mapped Cache:* O bloco é colocado aleatoriamente em qualquer posição, o que é pouco eficiente;

- *First In, First Out* (FIFO): Com essa estratégia o bloco será posicionado no lugar do bloco a mais tempo armazenado;

- *Least Recently Used* (LRU): O bloco a ser substituído será o que estiver a mais tempo sem ser utilizado.

- *Modified* LRU *Replacement:* Nessa estratégia para cada bloco será atribuída uma prioridade, alta ou baixa. Se o bloco for de baixa prioridade, ele só poderá substituir um bloco com a mesma prioridade. Caso tenha alta prioridade, ele poderá substituir qualquer bloco usando a estratégia LRU.

No Blackfin existem dois tipos de memória *cache*: de dados e de instruções que estão posicionadas na memória interna (L1) do processador e podem ou não serem configuradas como tal.

4.2.1.1 Memória *Cache* L1 de Instruções

A memória L1 de instruções é composta da combinação de bancos de memória que podem ser configurados como SRAM ou *cache*. A configuração dessa região como *cache* é feita através do registrador **IMEM_CONTROL**.

Quando habilitada como *cache*, a memória L1 de instruções trabalha como uma memória 4-way set associativa onde cada um dos sub-blocos pode ser bloqueado independentemente. O controlador de *cache* de instruções pode ser configurado para usar a estratégia de repreenchimento LRU ou LRU modificado.

O arranjo da memória *cache* de instruções é feita em quatro sub-bancos, que por sua vez, são compostos de 32 conjuntos e, que por sua vez, são compostos por 4 blocos. O tamanho de cada bloco é de 32 *bytes.* A Figura 4.1mostra essa configuração.

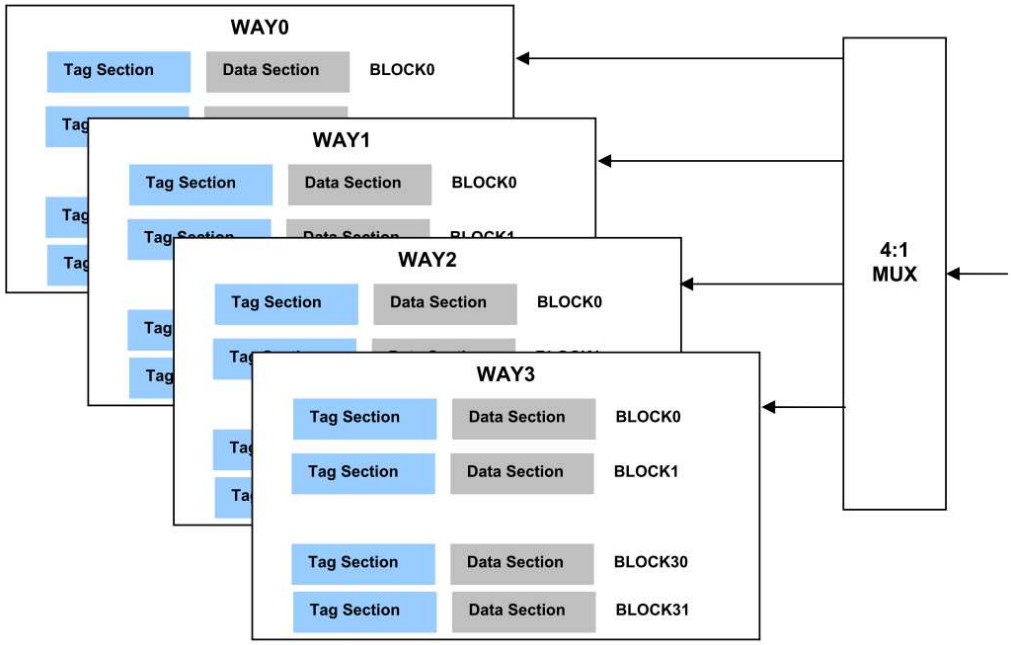


Figura 4.1 - Configuração da *cache* de instruções no Blackfin [21].

4.2.1.2 Memória *Cache* L1 de Dados

A *cache* L1 de dados também é do tipo *on-chip* e possui 32 Kbytes que podem ser configurados pelo registrador **DMEM_CONTROL** como SRAM ou *cache*. Essa memória é representada por dois bancos independentes de 16 Kbytes.De acordo com [21], a configuração dessa memória *cache* é do tipo 2-way set associativa, usando para repreenchimento a política LRU. Cada banco de 16 Kbytes é arranjado como 4 sub-bancos, que por sua vez,são compostos de 64 conjuntos, e cada conjunto tem 4 blocos de 32 *bytes*cada um. O esquema de arranjo da memória de dados é mostrado na Figura 4.2.

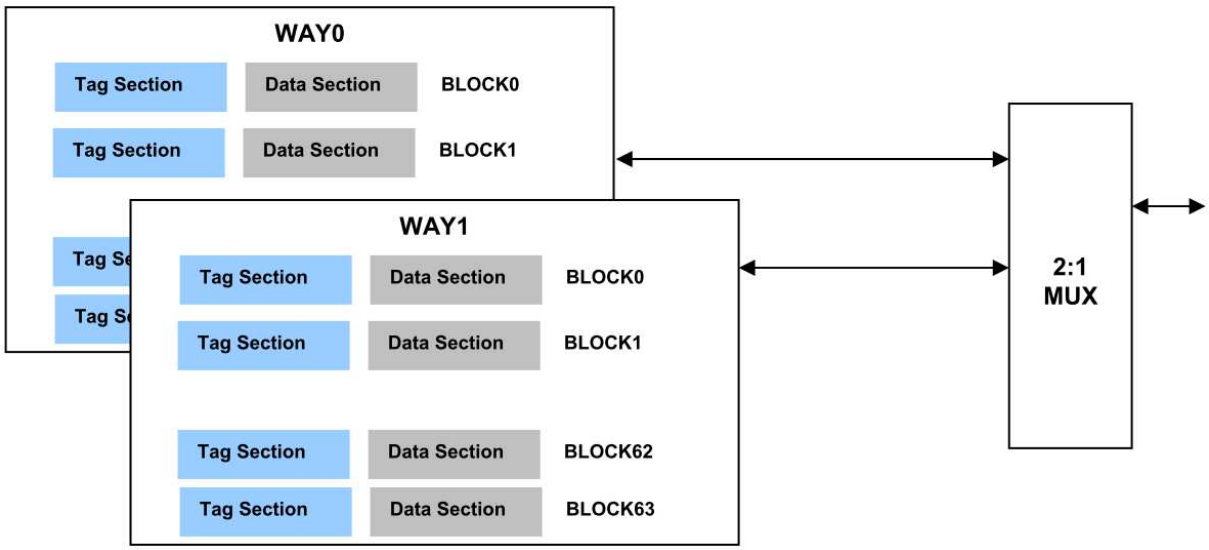


Figura 4.2 - Configuração da *cache* de dados do Blackfin [21].

4.2.2 Proteção de Memória

Para fornecer controle sobre a cacheabilidade de certas regiões de memória e gerenciamento dos atributos de proteção em nível de página, o Blackfin possui uma unidade de *Memory Management Unit* (MMU). Segundo [22], essa unidade fornece grande flexibilidade na alocação de memória e recursos de entrada/saída entre tarefas, com controle completo sobre os direitos de acesso e do comportamento da *cache*.

A MMU é implementada como dois blocos de 16 entradas de *Content Addressable Memory* (CAM). Cada entrada é referida como um descritor *Cacheability Protection Lookaside Buffer* (CPLB).

De acordo com [21], quando for ativada, cada entrada válida no MMU é examinada em qualquer operação de busca, carga ou armazenamento para determinar se existe uma correspondência entre o endereço que está sendo solicitado e a página descrita pela entrada do CPLB. Se ocorrer uma correspondência, a cacheabilidade e atributos de proteção contidos no descritor são usados para a operação de memória sem somar novos ciclos à execução da instrução.

4.2.3 Uso Eficiente de Memória

Como pode ser visto anteriormente, a estrutura da hierarquia de memória nos DSPs Blackfin possui um grande poder de armazenamento e um sistema de comunicação interna e externa muito eficaz. Mas isso se torna insignificante se não for trabalhado de forma apropriada. Com esse pensamento e considerando a complexidade e o tamanho da aplicação a ser desenvolvida a preocupação com este fator se torna essencial.

Para esse gerenciamento de memória, o desenvolvedor pode atuar juntamente com o compilador para alcançar os resultados pretendidos. Para isso, as aplicações desenvolvidas no VisualDSP++ geram um arquivo que permite ao programador controlar o posicionamento dos dados em memória. Esse arquivo é o *Linker Description File* (LDF).

Como também foi descrito anteriormente, a memória de dados é dividida em seções e o uso adequado dessas seções é um dos responsáveis pela alocação eficiente. Assim, colocar conjuntos de dados em diferentes seções de memória pode ajudar no desempenho.Isso só é possível porque o *hardware* do processador pode suportar duas operações de memória em uma simples linha de instrução. Mas somente será possível se os dois endereços estiverem situados em bancos de memória diferentes. Caso estejam no mesmo banco o uso dessa característica não mais será possível.

Então cabe ao desenvolvedor "forçar" a colocação desses dados em seções distintas. Isso pode ser feito através da já citada diretiva de compilação `different_banks`ou de forma manual. Um exemplo de definições manuais de seções para conjunto de dados é mostrado na Figura 4.3.

```
1  void no_different_banks(void)
2  {
3
4  /* Código sem especificação de banco */
5
6      short a[100];
7      short b[100];
8      int i, sum = 0;
9
10     for(i=0;i<100;i++)
11     {
12         sum =+ a[i] * b[i];
13     }
14 }
15
```

```
1  void different_banks(void)
2  {
3
4  /* Código com especificação de banco */
5
6      section("L1_data_a");
7      short a[100];
8      section("L1_data_b");
9      short b[100];
10     int i, sum = 0;
11
12     for(i=0;i<100;i++)
13     {
14         sum =+ a[i] * b[i];
15     }
16 }
17
```

Figura 4.3 – Exemplo de implementação de diferentes bancos de memória.

A colocação manual de variáveis ou trechos de códigos em seções diferentes não está relacionada somente à possibilidade de vários acessos simultâneos à memória. Pode também ser usada para a definição de partes mais ou menos prioritárias.Em um código, por exemplo, existem funções que devido à sua complexidade ou devido sua importância necessitam de uma atenção especial para não comprometer o desempenho. Essa atenção especial pode ser feita através da definição de seções mais ou menos prioritárias.

Tabela 4.1 - Algumas partes do código e suas seções de memória correspondentes.

Parte do Código	Seção
`Open the AAC file`	`section("slow_noprio_code")`
`scaledFFT`	`section("fast_prio0_code")`
`cplxSynthesisQmfFiltering`	`section("fast_prio1_code")`
`decodeEnvelope`	`section("fast_prio2_code")`

A Tabela 4.1mostra alguns exemplos da aplicação de prioridades a certas funções que fazem parte do código do decodificador de áudio HE-AAC v2. Esse processo é feito por meio de definições de regiões de memória. As funções mais prioritárias são colocadas em seções normalmente dentro da memória interna do processador, onde haverá uma maior agilidade para alcançar esses dados.

Um exemplo de função de prioridade máxima para o decodificador de áudio HE-AAC v2 é a `scaledFFT`, que por sua prioridade foi colocada na seção mais rápida (`fast_prio0_code`). Outras funções que também são importantes, mas que não representam pontos chaves do código, são colocadas normalmente em seções mistas compostas por memória interna e memória externa, como o caso de `fast_prio1_code` e `fast_prio2_code`.

Por outro lado, algumas funções podem ser alocadas em seções localizadas somente em memória externa e isto não representará alterações no desempenho do código. Esse é o caso, por exemplo, da abertura do arquivo AAC que é alocada na seção `slow_noprio_code`.

4.2.4 Configuração da Razão entre CCLK/SCLK

Uma das considerações importantes que devem ser feitas para obter uma aplicação eficiente é em relação aos valores de*clock* no Blackfin. O *clock* é um sinal de referência que é

utilizado para coordenação das ações de diversos dispositivos. Então, se a magnitude desse sinal for maior ou menor afetará inversamente o tempo de execução de um código, segundo a relação tempo-frequência.

Nos DSPs Blackfin existem dois valores de referências, o *Core Clock* (CCLK) que é responsável pela sincronização dos eventos que acontecem no núcleo do processador (*on-chip*) e o *System Clock* (SCLK) que é responsável pelo sincronismo da comunicação entre os periféricos.

Esses dois valores são derivados do sinal de *clock* de entrada (CLKIN) que fornece a frequência de *clock* necessária, o ciclo de trabalho e a estabilidade para permitir a exata multiplicação do *clock* interno por meio do módulo *on-chip Phase Locked Loop* (PLL).

Durante a operação normal, o usuário programa o PLL com um fator de multiplicação para CLKIN. O sinal resultante da multiplicação é conhecido como *Voltage Controlled Oscillator*(VCO), que será usado para gerar o valor dos *clocks* conforme descrito em[15].

No caso específico do Blackfin ADSP-BF527, o CLKIN corresponde a 25 MHz, assim o VCO será obtido por meio da multiplicação do CLKIN por um fator que varia de 0,5x até 64x, tendo como valores mínimos e máximos um VCO de 12,5 e 1600 MHz. Esse valor máximo para o processador em questão é inatingível, pois a frequência máxima de trabalho está limitada a 600 MHz. O valor padrão usado para esse DSP é obtido pela multiplicação do CLKIN por 10 (VCO = 250 MHz).

De posse do VCO, agora é possível calcular os valores de CCLK e SCLK. Para a alteração destes valores, o desenvolvedor deve usar os registradores **PLL_CTL** e **PLL_DIV**, respectivamente.Alcançar uma boa relação entre os dois valores de *clock* que são usados no DSP é de fundamental importância para que haja uma eficiência tanto no processamento interno quanto na comunicação utilizando os barramentos. O ideal seria que esses dois *clocks* trabalhassem em suas potências máximas para que não haja atrasos excessivos e consequentemente dificuldades na comunicação, mas isso não é possível.

As Tabelas 4.2 e 4.3 mostram as possíveis relações entre o valor VCO e os valores de CCLK e SCLK.

Tabela 4.2 - Razão do *core clock*[15].

Nome do Sinal	Razão do Divisor VCO/CCLK	Exemplos de Razões de Frequências (MHz)	
		VCO	CCLK
00	1	300	300
01	2	600	300
10	4	600	150
11	8	400	50

Tabela 4.3 - Razão do *system clock*[15].

Nome do Sinal	Razão do Divisor VCO/SCLK	Exemplos de Razões de Frequências (MHz)	
		VCO	SCLK
0000	Reservado	-	-
0001	1:1	100	100
0010	2:1	200	100
0011	3:1	400	133
0100	4:1	500	125
0101	5:1	600	120
0110	6:1	600	100
N = 7 - 15	N:1	600	600/N

Outro ponto a ser considerado quando há o intuito de se obter o melhor desempenho do sistema é a relação entre o CCLK e o SCLK. Essa razão, segundo [23], afeta a taxa de transferência do sistema, pois com o valor do*clock* do núcleo baixo, sincronismo e latências aumentadas dos barramentos do núcleo resultam em uma utilização reduzida do barramento do sistema.

Aumentar muito somente um dos *clocks* ou definir valores semelhantes para os dois *clocks* podem não representar melhorias no sistema. Um exemplo pode ser visto na Tabela 4.4que mostra os resultados para diferentes razões de CCLK/SCLK para um teste onde um simples canal de DMA transfere um *buffer* de 8 Kbytes entre a memória L1 e L2.

Tabela 4.4 - Efeitos da razão CCLK/SCLK na taxa de transferência [23].

Acesso DMA	Razão CCLK/SCLK	Transferências por Segundo
Escrita	5	57303
	4	46265
	3	46265
	2	33301
	1	23378
Leitura	5	51165
	4	51165
	3	51165
	2	38757
	1	29196

Como pode ser visto na Tabela 4.4, os resultados a partir da taxa 2:1 caem consideravelmente. A leitura é menos sensível, já que as latências dos barramentos do núcleo são escondidas na latência entre o tempo que o controlador de memória informa a solicitação e o momento que os dados estão disponíveis.

4.3 OTIMIZAÇÃO EM RELAÇÃO AO COMPILADOR

Uma das possibilidades para se obtermelhorias significativas é por meio da otimização do código-fonte para melhorar o uso do compilador. Oscódigos-fontes das aplicações para processadoresBlackfin podem ser escritos tanto em linguagem *assembly*, assim como em C/C++, mas normalmente sãoescritos em linguagem do padrão ANSI C, conforme[24].

A linguagem C é utilizada principalmente por ser uma linguagem de programação que requer poucos recursos,tanto para odesenvolvimento quanto para a manutenção. No entanto ela tem algunspontos que não são ideais para esse tipo de utilização, principalmente o fato de não ter sido desenvolvidapara o processamento de sinais e não ter uma vocação matemática muito forte. Outro fator negativo está no fato de os DSPs “esperarem” o uso de linguagem *assembly* em áreastidas como críticas.

Por todas essas razões o desempenho de um código compilado que foi escrito em linguagem C sem preocupações com questões de otimização deve ser muito menor caso tivesse sido escrito em linguagem *assembly.*

4.3.1 O Ambiente de Desenvolvimento no Auxílio da Otimização

Para iniciar a otimização,o VisualDSP++ dispõe de algumas ferramentas que auxiliam o desenvolvedor na tomada de decisões na otimização de um código.

O primeiro passo a ser tomado para a otimização de um código é entender onde estão os pontos que mais necessitam serem otimizados. Para isso o VisualDSP++ disponibiliza uma ferramenta que auxilia o desenvolvedor: o *Statistical/Linear Profiler.*

A diferença entre o *Statistical* e o *LinearProfiler* é basicamente a forma de comunicação com o DSP. Quando se encontra em ambiente de simulação, utiliza-se o *Linear Profiler* e nos demais tipos de comunicação estará presente o *Statistical Profiler.* A Figura 4.4mostra um exemplo típico do *Statistical Profiler.*

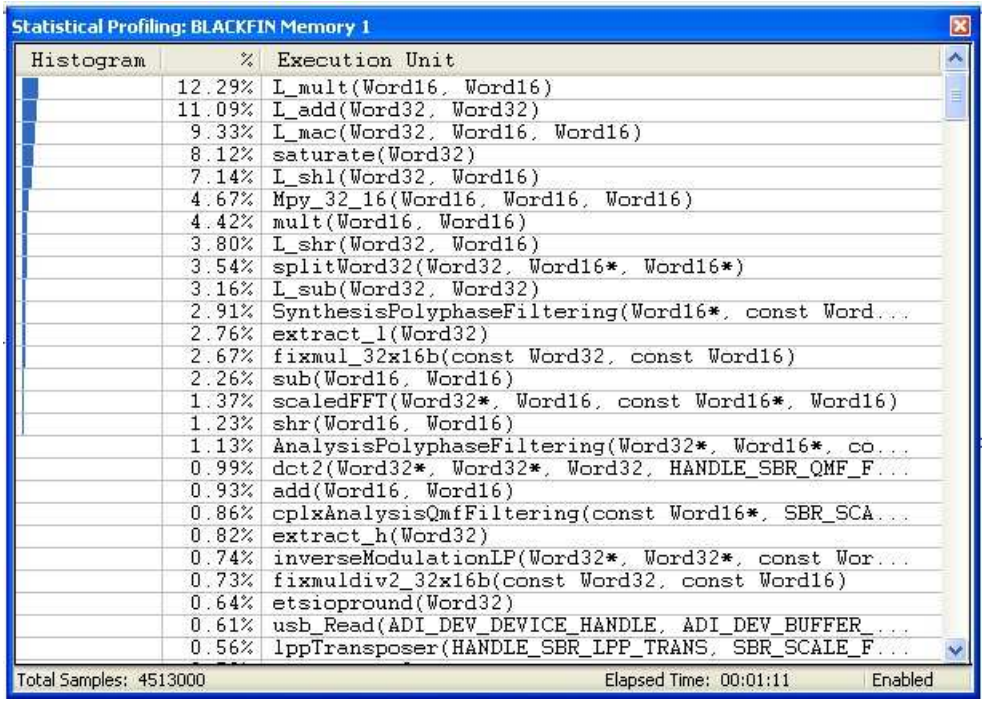


Figura 4.4 - Janela do *Statistical Profiler*.

Enquanto o programa está sendo executado, o *Statistical Profiler* é atualizado periodicamente com o valor do contador do processador que, associado com os valores locais do programa, apresenta a porcentagem de tempo de execução gasto em cada função.

AFigura 4.4ilustra essa ferramenta. Nela há um histograma onde é possível ver a porcentagem, além da quantidade de ciclos gastos na execução de uma função. A partir da coleta destes dados é possível direcionar o trabalho de otimização.

Outra ferramenta bastante útil e que facilita o trabalho do desenvolvedor é o *Disassembly*. Essa janela, de acordo com [17], mostra o código fonte de forma "desmontada", o que é muito útil para modificações temporárias no código para visualização quando o código fonte não está disponível, por exemplo. Permite examinar o código *assembly* gerado pelo compilador C/C++.

A Figura 4.5mostra um trecho de código escrito em linguagem C e o código gerado pelo *Disassembly*.

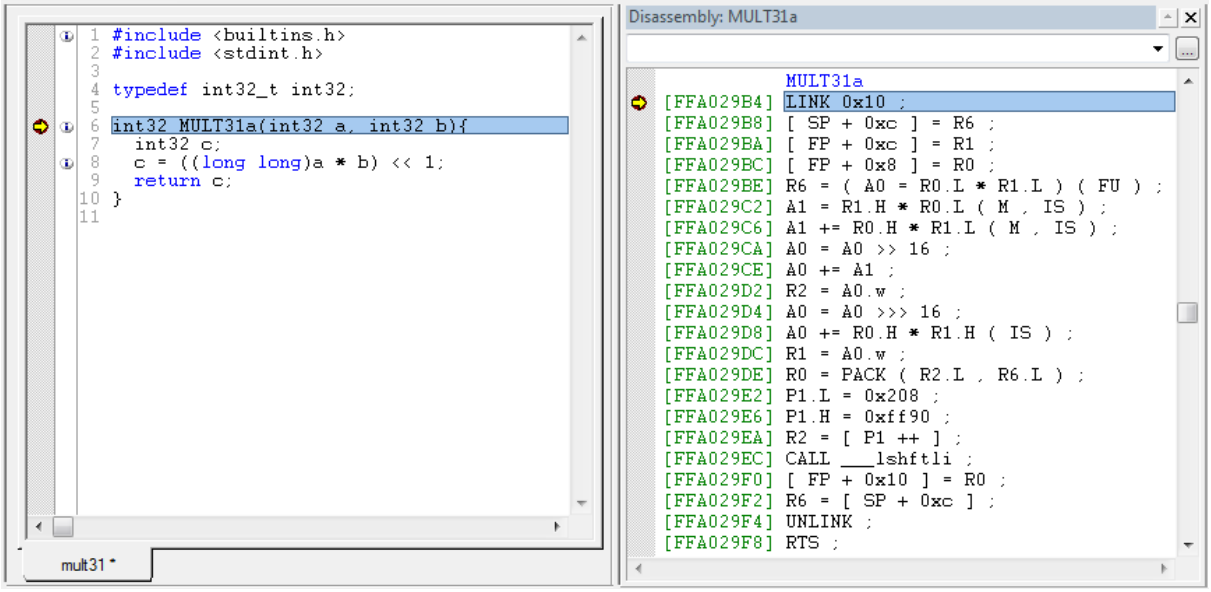


Figura 4.5 - Trecho de código em C e seu correspondente no *Disassembly*.

Essa é também uma ferramenta muito importante ao desenvolvedor, pois dá a possibilidade de saber como o compilador está gerando o código *assembly* que será usado no DSP. A partir desses dados muitas vezes é possível encontrar situações onde, por exemplo, o compilador gera um código incorreto ou com possibilidades óbvias de otimização.

Há também outra opção de otimização dada pelo VisualDSP++ que é a otimização do compilador. Essa opção pode ser habilitada através da opção de compilação "-O" ou através da janela de opções do projeto como pode ser visto em Figura 4.6.

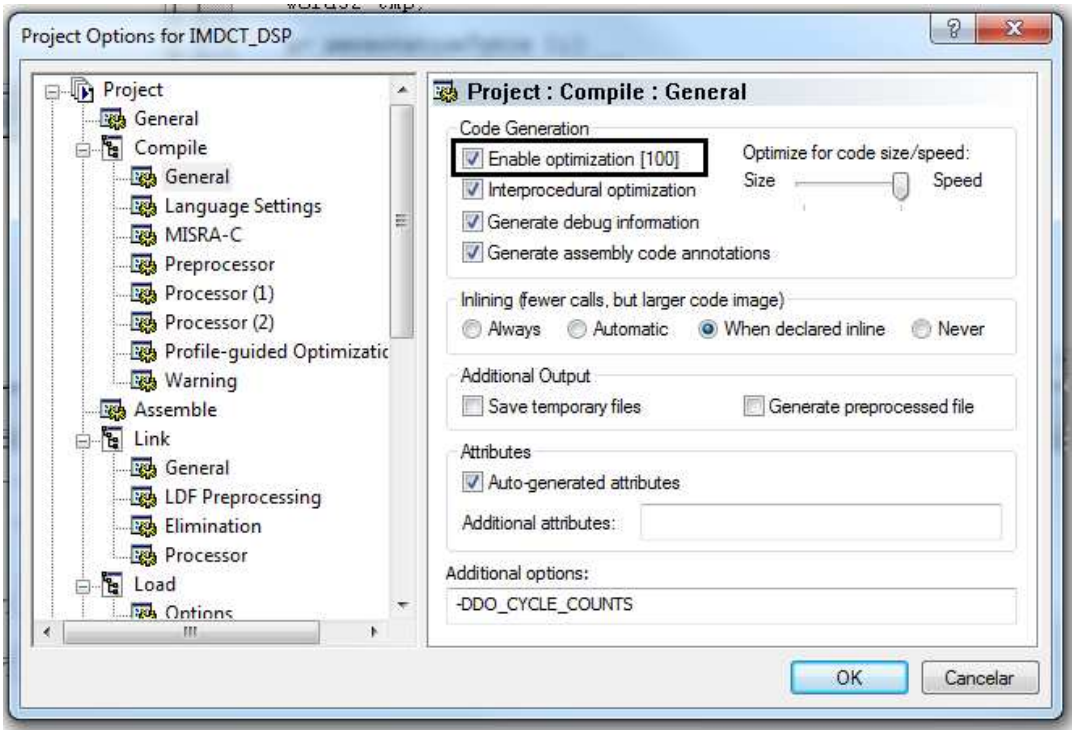


Figura 4.6 - Opção de otimização do compilador nas opções do projeto.

O otimizador do processador Blackfin gera um código eficiente, em tempo de execução, do código original que foi escrito,sem se preocupar com questões de otimização. Para o desenvolvedor não haverá mudanças visuais no código, mas o compilador passará a analisá-lo de forma diferente e consequentemente gerará um código *assembly* bem mais otimizado, podendo obter, segundo[17], códigos até vinte vezes mais rápidos.

O VisualDSP++ disponibiliza diversas outras ferramentas que irão auxiliar o desenvolvedor na busca de um código com um melhor desempenho. Outras ferramentas serão apresentadas no decorrer do texto para auxiliar na aplicação de outras técnicas de otimização.

4.3.2 Tipo de Dados

A escolha do tipo de dados a ser utilizado é algo muito importante. O conhecimento de como o compilador vai implementar o código terá um efeito significativo sobre a eficiência do código compilado.

A linguagem C define vários tipos de dados, mas não o tamanho deles, o que pode variar de um processador para outro. Por isso, é fundamental conhecer quais são os tipos de dados suportados pelo processador para gerar um código otimizado.

Os DSPs Blackfin suportam palavras de 16 ou 32 *bits* e *bytes*. As palavras de 16 e 32 *bits* podem ser inteiras ou fracionárias, mas *bytes* são sempre inteiros. Tipos de dados inteiros podem ser sinalizados ou não-sinalizados, mas tipos de dados fracionários são sempre sinalizados[15].

Mesmo sendo concebidos para operarem cálculos em aritmética de ponto-fixo, o Blackfin também pode emular operações com ponto flutuante em *software* de acordo com o padrão IEEE-754[25].

Para exemplificar o fato de que a escolha do tipo de dado correto influencia no desempenho da aplicação foram desenvolvidas duas aplicações semelhantes mas escritas em aritméticas diferentes e,em seguida, foram comparados seus resultados. As duas aplicações executam uma parte do decodificador de áudio HE-AAC v2que representa grande parcela da complexidade total do decodificador, a *Inverse Modified Discrete Cosine Transform* (IMDCT). A comparação entre estas duas versões da IMDCT é mostrada na Tabela 4.5.

Tabela 4.5 - Comparação entre implementação em ponto-fixo e ponto-flutuante no Blackfin

Bloco	Ciclos
IMDCT *Fixed*	115.080
IMDCT *Float*	348.530

Pelos dados obtidos por meio da Tabela 4.5 pode-se confirmar que a utilização do tipo de dado correto tem grande influência no desempenho. Mesmo o Blackfin tendo condições de tratar dados em ponto-flutuante, o fato dessa aritmética não ser nativa e sim emulada em *software* fez que o desempenho fosse mais de 200% inferior ao ponto-fixo.

4.3.3 Funções Intrínsecas

Como pode ser visto anteriormente, o Blackfin não suporta apenas dados inteiros, também suporta dados fracionários do tipo `fract16` e `fract32`. Para se trabalhar com dados fracionários existem duas possibilidades. A primeira delas envolve o uso de longos deslocamentos promovidos por *shifts* e multiplicações. A segunda forma envolve o uso de

funções intrínsecas do compilador.

As funções intrínsecas (ou *bult-in functions*) permitem o uso eficiente de características de baixonível do *hardware* do processador durante a programação em alto nível. Assim podem ser acessados recursos do processador de forma otimizada sem a necessidade de programação em linguagem *assembly*.

Existem vários grupos de funções intrínsecas no Blackfin que são encarregados de tipos específicos de operações. Dentre os diversos grupos descreve-se aqui mais detalhadamente dois deles: *Fractional Value Built-In Functions in* Ce ETSI *Support.*

4.3.3.1 *Fractional Value Built-In Functions in* C

Como já foi dito, uma das formas de se tratar dados fracionários é por meio de funções intrínsecas. Para isso foi criado o *fractional value built-in functions in* C que é um grupo de funções que fornecem acesso à aritmética fracionária e operações paralelas de 16 *bits* suportadas pelo processador de instruções do Blackfin.

Como a aritmética fracionária trabalha com instruções um pouco diferentes da aritmética inteira não é recomendável que se use operadores padrões do C em dados fracionários. Isso pode fazer com que os resultados gerados não sejam corretos. Assim, as funções intrínsecas são a opção correta para se trabalhar com dados fracionários.

Existem várias funções que tratam de forma otimizada estes dados. Essas funções são declaradas no arquivo `fract.h`e são separadas em três grupos: `_fr1x16, _fr2x16`e `_fr1x32`que agrupam funções com dados do tipo `fract16, double fract16`, e `fract32` respectivamente.

Dentre as várias funções pertencentes a este grupo, serão citadas apenas algumas para exemplificar as principais operações matemáticas presentes[26].

- `fract16 add_fr1x16(fract16 f1, fract16 f2)` – Realiza a adição de duas variáveis de 16 *bits*.
- `fract32 mult_fr1x32(fract16 f1, fract16 f2)` – Realiza a multiplicação fracionária de dois fracionários de 16 *bits*, retornando um resultado de 32 *bits*.
- `fract32 mult_fr1x32x32(fract32 f1, fract32 f2)`– Realiza a multiplicação de duas variáveis de 32 *bits*. O resultado é arredondado para 32 *bits*.

−`fract32 shr_fr1x32(fract32 src, short shft)`– *Shift* aritmético da variável `src` para a direita em `shft`com extensão de sinal. Se `shft`for negativo, o *shift* será para a esquerda com o valor absoluto de `shft.`

4.3.3.2 ETSI *Support*

De forma semelhante ao anterior, este é um grupo que manipula os tipos de dados `fract16` e `fract32`. Desenvolvido pelo *European Telecommunications Standards Institute* (ETSI) e definido no arquivo `libetsi.h`trata-se de um conjunto de funções intrínsecasque podem ser utilizados também no Blackfin.

Dentre as muitas funções deste grupo, podem serão citadas algumas delas[26]:

− `fract16 add(fract16 f1, fract16 f2)`– Retorna a soma de duas variáveis de 16 *bits* em um resultado também de 16 *bits*.

− `fract32 L_mult(fract16 f1, fract16 f2)`– Retorna o resultado de 32 *bits* com saturação da multiplicação de duas variáveis fracionárias de 16 *bits*.

− `fract32 Mpy_32(fract16 hi1, fract16 lo1, fract16 hi2, fract16 lo2)` – Realiza a multiplicação de dois valores de 32 *bits* já decompostos em parte alta e parte baixa.

−`fract32 L_shr(fract32 src, fract16 shft)`– Realiza o *shift* aritmético de 32 *bits* para a direita pelo valor dado em `shft.`Se `src` for negativo o *shift* será para a esquerda.

Independente de qual grupo de funções intrínsecas for escolhido estas funções têm resultados,no Blackfin, superiores às funções convencionais. A Tabela 4.6 mostra algumas comparações feitas entre esses tipos de funções.

Tabela 4.6 - Comparação de diferentes formas de implementação de uma mesma função

Função	Técnica de Implementação			Percentual de Melhoria	
	Normal (Ciclos)	ETSI (Ciclos)	Fract Built-In (Ciclos)	Melhoria ETSI	Melhoria Fract Built-In
Adição (fract32 , fract32)	54	21	19	61.1%	64.8%
Multiplicação (fract16 , fract16)	35	21	19	40.0%	45.7%
Shift Left (fract32 , fract16)	122	28	22	77.0%	81.9%

Na Tabela 4.6 foram implementadas algumas funções de três formas diferentes: implementação normal, implementação usando as funções intrínsecas do grupo ETSI e implementação usando as funções intrínsecas do grupo *Fract Value Built-in Functions.* Em cada função além da operação matemática descrita, foram feitas verificações de *overflow* e saturação para garantir um resultado condizente.

Como pode ser visto na tabela anterior, o uso de funções intrínsecas representam uma melhora no desempenho de até 81.9% se comparado à implementação normal. Isso reforça mais uma vez o quanto que ter conhecimento acerca do funcionamento do DSP interfere na produção de um código otimizado.

Pode ser notada também por meio da Tabela 4.6 uma superioridade das funções Fract Built-*In* sobre as funções ETSI. Isso mostra que mesmo semelhantes e tratando as operações de forma otimizada, o primeiro grupo possui um nível de otimização ainda maior por se tratar de funções desenvolvidas especificamente para a arquitetura do Blackfin, o que as credenciam para serem utilizadas neste trabalho.

4.3.4 Funções Inline

Os processadores da família Blackfin,por meio do VisualDSP++, suportam o uso de código *assembly*integrado, ou seja, permitem que uma pequena seção de código *assembly*seja inserida dentro de um programa escrito em C.

Essa é uma boa maneira de acessar os recursos especializados do processador e será executada mais rapidamente do que uma chamada a uma função separada escrita em *assembly*,

pois elimina a sobrecarga da chamada de funções aumentando assim a velocidade de execução do programa.

O uso de funções inlineé definido pela palavra-chave `inline`no cabeçalho da função. Isso indica que a função deve ter o código C gerado de forma integrada no momento da chamada.

Mesmo sendo responsável por um melhor desempenho do código essa palavra-chave somente será útil para funções pequenas e constantemente usadas, pois o ganho na velocidade de execução vem acompanhado em um pequeno incremento no tamanho do código. Uma recomendação é evitar seu uso juntamente com funções intrínsecas, pois afetará o bom funcionamento de ambas[14].

Para exemplificar o uso de inline, a Figura 4.7 mostra um exemplo de função declarada como inline e a Tabela 4.7 mostra a comparação entre a implementação dessa função com uma função sem inline.

```
1
2 inline int sim_inline(int a, int b){
3     return (a*b);
4 }
5
```

Figura 4.7 - Implementação simples de uma função de multiplicação com o uso de inline.

Tabela 4.7 – Comparação entre implementação da função anterior sem e com o uso de inline.

Funções	Ciclos
Função sem Inline	36
Função com Inline	15

Como pode ser visto na Tabela 4.7, o uso da funçãoinline representa uma melhoria de 58.3% se comparada com a implementação da mesma função sem o uso de inline.

4.3.5Pragmas

Uma forma de melhorar a aplicação, suportada pelo Blackfin, é utilizando pragmas. Os pragmas são diretivas de implementação específicas que modificam o comportamento do compilador. Dentre as possíveis mudanças de comportamento uma delas está relacionada ao nível de otimização. A seguir serão descritos alguns desses pragmas e seu funcionamento.

4.3.5.1 Pragmas de Alinhamento de Dados

Esse tipo de pragma é utilizado para modificar a forma com que o compilador organizará os dados na memória. A arquitetura do processador Blackfin requer que os acessos à memória estejam alinhados, isto é, que independente do tamanho do bloco que será colocado na memória o endereço onde o bloco será armazenado terá que ser múltiplo do tamanho do tamanho da palavra [26]. Por exemplo, dados do tipo `short`de dois *bytes* tem um alinhamento de 2. Um dado do tipo `long`de quatro *bytes* tem um alinhamento de 4, ou seja, só serão colocados em endereços múltiplos de quatro.

O maior problema se dá no alinhamento de estruturas, pois nem sempre o alinhamento do primeiro membro resultará no alinhamento automático dos demais.Na definição de uma `struct`o alinhamento global será definido de acordo com a variável de maior alinhamento, o que pode gerar entre outras coisas, um acréscimo no tamanho necessário para armazenamento desta estrutura em disco. Isso pode representar um problema em aplicações com grande número de variáveis. Dentre os principais pragmas de alinhamento podem ser citados[26]:

- `#pragma align` *num*– obriga a próxima variável ou campo de declaração a forçar o alinhamento em um limite especificado pela variável `num;`

- `#pragma alignment_region` – obriga que não só uma variável, mas sim um grupo de dados consecutivos seja alinhado;

Para demonstrar como os pragmas de alinhamento de dados realmente trazem melhorias, na Figura 4.8são demonstradas duas estruturas semelhantes, sendo que a estrutura da esquerda está desalinhada e a estrutura da direita está alinhada pelo uso de pragmas.

```
1  /* Estrutura Sem Dados Alinhados */
2  typedef  struct   {
3                char   c1;
4                short  s;
5                char   c2;
6                float  f;
7           } tEstrutura2;
8
```

```
1  /* Estrutura Com Dados Alinhados */
2  typedef  struct   {
3              #pragma align 1
4              char   c1;
5              #pragma align 1
6              short  s;
7              #pragma align 1
8              char   c2;
9              #pragma align 1
10             float  f;
11
12          } tEstrutura1;
13
```

Figura 4.8 - Exemplo do uso de pragmas para alinhamento de dados.

Antes de ser alinhada a estrutura tinha um tamanho de 12 *bytes*, ou seja, devido ao desalinhamento existente, eram desperdiçados 4 *bytes*. Já na estrutura alinhada estes 4 *bytes* desperdiçados são removidos, gerando o melhor alinhamento possível dos dados da estrutura.

4.3.5.2 Pragmas de Otimização Geral

Existem vários tipos de pragmas com capacidade de alterar o nível de otimização de um módulo durante sua compilação. Para usá-los é necessário que sejam utilizados imediatamente antes da definição da função a ser otimizada e que sejam globais. Dentre esses pragmas podem ser mencionados [26]:

- `#pragma optimize_off`– desabilita o otimizador, se estiver habilitado. Ele temo mesmo efeito da compilação sem nenhuma otimização;

- `#pragma optimize_for_space`– habilita o otimizador, se estiver desabilitado,ou define o foco para redução do tamanho do código na otimização.

- `#pragma optimize_for_speed`– habilita o otimizador, se estiver desabilitado, ou define o foco da otimização para obter um alto desempenho.

Esta classe de diretivas de otimização trabalha de forma semelhante à opção "habilitar otimização" do VisualDSP++, que pode ser visto na Figura 4.6. A diferença é que o uso dos pragmas dá ao desenvolvedor uma maior liberdade de otimizar o código de forma mista, podendo realizar as alterações que julgue ser positivas.

O uso do pragma para otimização de espaço, por exemplo, é recomendável em funções que raramente serão chamadas, o pragma para otimização de velocidade é recomendável em funções críticas para o desempenho,e o pragma para desabilitar a otimização é útil para depuração de funções específicas sem aumentar o tamanho ou diminuir o desempenho da aplicação de forma desnecessária.

4.3.5.3 Pragmas de Otimização de *Loop*

A otimização dos *loops* é normalmente a parte onde a aplicação de pragmas é mais eficaz [14]. Esses pragmas são colocados imediatamente antes da instrução de *loop* para fornecer ao compilador a informação de que nessa parte do código é possível realizar uma otimização

mais profunda. Podem ser mencionados como pragmas de otimização de *loops*:

- `#pragma loop_count` – informa ao compilador por meio de seus parâmetros *min, max* e *modulo* a quantidade mínima e máxima de vezes que o *loop* será executado e se o contador do *loop* é múltiplo de uma constante. Sabendo dessas informações o compilador estará habilitado a tomar decisões sobre a técnica de otimização mais adequada para o laço.

- `#pragma no_vectorization`– informa ao compilador que no *loop* imediatamente a seguir não será necessária a aplicação da vetorização, ou seja, da execução de mais de uma iteração do *loop* em paralelo. Passar esse tipo de informação ao compilador pode ser importante, por exemplo, em *loops* com poucas iterações onde a vetorização pode não ser muito vantajosa.

- `#pragma vector_for`– informa ao compilador que é seguro executar duas iterações do *loop* em paralelo. Em outras palavras, diz ao compilador que a execução de duas iterações em paralelo não afetará o resultado final do *loop*. Esse pragma não força a otimização do *loop*. A decisão final será do compilador que se baseará em várias propriedades, dentre elas as informações do pragma.

- `#pragma different_banks`– permite que o compilador assuma que grupos de dados utilizados no *loop* podem ser declarados em bancos de memória diferentes. Isso faz com que dois acessos independentes à memória podem ser realizados em conjunto, sem acontecer nenhum tipo de perda.

A otimização de *loops* é um campo que vai muito além da utilização de pragmas. Outros aspectos dessa otimização serão descritos a seguir.

4.3.6Otimização de Loops

A otimização dos *loops*ou laços de repetição, como por exemplo, *for, while*, *do while*, é uma importante ferramenta para melhoria do desempenho do aplicativo. Primeiramente por ser algo que está presente em grande parte dos algoritmos da área de processamento digital de sinais e também porque qualquer ganho obtido nestas instruções será multiplicado pela quantidade de iterações que serão realizadas.

Alguns conceitos utilizados na otimização de *loops*foram descritos na seção que tratou do uso de pragmas para a melhoria destas estruturas. Outras ideias envolvidas nesse contexto serão descritas a seguir.

4.3.6.1 *Pipeline* em *Software*

Pipeline é uma técnica de *hardware* que permite que o processador inicie o processamento de uma instrução antes da instrução anterior ter sido completada. De forma análoga o *pipeline* realizado em *software* permite que o código gerado inicie o processamento de uma nova iteração do *loop* original antes do término da iteração anterior [26]. Esse processo é usado se uma parte do algoritmo não depender de outras.

O *pipeline* em *software*é usado para organizar um *loop* de modo que operações de iterações distintas sejam executadas de forma sobreposta, isto é, que sejam executadas ao mesmo tempo utilizando o paralelismo do processador. Com isso a ordem do *loop* será alterada de modo a realizar o máximo de instruções ao mesmo tempo, desde que não altere a semântica do código.

4.3.6.2 *Loop Rotation*

Para se realizar o *pipeline* em *software,* uma técnica que contribui para isso é o *loop rotation*. Essa técnica faz, como o próprio nome sugere, a rotação das instruções do *loop* alterando o começo lógico e as posições finais do laço para permitir um *loop* mais eficiente. Um exemplo de implementação desta técnica pode ser visto na Figura 4.9.

```
1   int no_loop_rotation(void)
2   {
3   /* Sem Loop Rotation */
4
5
6       for(i=0;i<10;i++)
7       {
8           a[i]=i*10;
9           b[i]=i*11;
10          c[i]=i*12;
11          d[i]=i*13;
12          e[i]=i*14
13      }
```

```
1   int loop_rotation(void)
2   {
3   /* Com Loop Rotation */
4       a[i]=0*10;
5       b[i]=0*11;
6       c[i]=0*12;
7
8       for(i=1;i<9;i++)
9       {
10          d[i-1]=(i-1)*13;
11          e[i-1]=(i-1)*14;
12          a[i]=i*10;
13          b[i]=i*11;
14          c[i]=i*12;
15      }
16
17      d[i]=9*13;
18      e[i]=9*14;
```

Figura 4.9 - Exemplo de aplicação da técnica de *loop rotation.*

A parte esquerda da Figura 4.9 representa um *loop* tradicional e a parte direita representa o mesmo *loop* só que desta vez rotacionado. Pode-se notar que a ordem das instruções a serem executadas é a mesma, o que difere é a quantidade de iterações a serem realizadas. Enquanto na versão original eram necessárias N iterações na versão otimizada este número foi reduzido para N-1.

No que se refere ao ganho de desempenho, essa técnica também traz bons resultados. Enquanto na implementação original eram necessários 667 ciclos, na versão otimizada são necessários apenas 544 ciclos, o que representa um decréscimo 18,5%.

4.3.6.3 *Loop Unrolling*

Outra técnica para otimização de laços de repetição que juntamente com *loop rotation* possibilitam o uso do *pipeline* em *software* é o *loop unrolling*. Através dessa técnica o programador reescreve o *loop* de forma a repetir explicitamente o corpo do laço, se atentando para o ajuste do índice.

Essa técnica pode ser implementada manualmente ou através do pragma `loop_unroll`.Como foi dito, os pragmas modificam o comportamento do compilador e neste caso específico modificam a forma com que o compilador otimizará este *loop*, fazendo que neste laço possam ser realizadas duas, ou mais iteraçõesem uma única execução sem prejuízos ao resultado das operações.

```
1  void no_loop_unrolling()
2  {
3
4  /* Sem Loop Unrolling */
5      for (i = 0; i < 10; ++i)
6      {
7        c[i] = b[i] + a[i];
8      }
9
```

```
1  void loop_unrolling()
2  {
3
4  /* Com Loop Unrolling */
5
6      short xa, xb, xc, ya, yb, yc;
7
8      for (i = 0; i < 10; i+=2) {
9        xb = b[i];
10       yb = b[i+1];
11       xa = a[i];
12       ya = a[i+1];
13       xc = xa + xb;
14       yc = ya + yb;
15       c[i] = xc;
16       c[i+1] = yc;
17     }
18
```

Figura 4.10 - Exemplo da implementação da técnica *loop unrolling*.

A Figura 4.10 apresenta um exemplo da aplicação da técnica *loop unrolling*. Na parte direita da figura, é mostrada a forma original do *loop* e na esquerda o laço depois de ser desenrolado. Neste caso específico o compilador foi orientado pelo pragma a realizar as operações referentes a dois valores de incrementosequenciais (i e $i+1$) em uma única execução do laço de repetição. A vantagem nesse caso também estará diretamente ligada à quantidade de iterações a serem executadas, que nesse caso diminuíram de N para N/2.

Essa diminuição do número de iterações pela metade não foi por acaso. O fato de se desenrolar o *loop* mais uma vez está diretamente relacionado à arquitetura do núcleo do DSP descrito na Figura 3.2, onde é possível notar que os recursos são duplicados.

A implementação desse exemplo trouxe os seguintes resultados. Enquanto a função original levava 292 ciclos para ser executado, depois de otimizada, seu número de ciclos foi reduzido para 265, representando uma melhora de pouco menos de 10%.

O que acontece no caso do *loopunrolling*e que pode ser considerado um problema é o fato de que em contrapartida a esse ganho no número de ciclos há um crescimento do tamanho do código. Esse crescimento leva o programador a aplicar essa técnica somente em funções que são consideradas pontos-chave do algoritmo quando há preocupações ou restrições de espaço.

Tanto a técnica *loop rotation* quanto a *loop unrolling*representam melhorias que trazem uma maior complexidade ao código, pois ele passa a ser mais difícil de ser interpretado e lido pelo desenvolvedor ou por alguém que venha a utilizá-lo.

Como o ganho de desempenho com o uso dessas técnicas é fundamental, para que o código esteja otimizado essa dificuldade de interpretação é desconsiderada.

4.3.7 Otimização de Baixo Nível

Após todas as propostas de otimização de algoritmo terem sido utilizadas e se ainda não foi alcançado o nível de otimização desejado, parte-se então para a aplicação de técnicas de otimização utilizando a linguagem *assembly.*

Esse tipo de otimização é considerado um método que traz melhorias consideráveis ao algoritmo, mas requer um nível de conhecimento do processador e da linguagem *assembly* muito maior, além de exigir um esforço em recursos humanos.

O que torna esse tipo de otimização tão vantajoso é o fato de por meio da linguagem

assembly o acesso aos recursos do DSP é feito de forma direta, evitando assim gastos desnecessários de muitos ciclos. Aqui, apenas para ilustração, é apresentado um exemplo de otimização em *assembly*querealiza um*loop* em *hardware.*

Ao contrário do *loop* realizado em *software* que faz o incremento/decremento de variáveis auxiliares e seguidas análises para a verificação da condição de parada, o *loop* realizado em *hardware* não possui nenhum gasto com operações que não sejam as utilizadas dentro do próprio *loop*.

Para demonstrar a utilização dessa ferramenta do processador Blackfin, o exemplo abaixo mostra um simples programa para ilustrar a utilização do *loop* em *hardware*.

```
P5 = 0X20;
LSETUP (loop_start, loop_end) LC0 = P5;
loop_start:
R2 = R0 + R1 || R3 = [P1++] || R4 = [I1++];
loop_end:
R2 = R3 + R4;
```

Nesse exemplo é demonstrada a implementação de um *loop* em *hardware* que executará as operações de `loop_start` e `loop_end`a quantidade de vezes descritas no registrador `P5`. Outra vantagem desse tipo de técnica é a possibilidade de se executar várias operações que não sejam concorrentes de forma paralela, gastando assim menos ciclos para ser executada.

É importante ainda salientar que a eficiência do código *assembly* gerado depende do nível de expertise do programador no DSP alvo.

Capítulo 5

A Implementação do Decodificador

Este capítulo aborda o desenvolvimento propriamente dito do decodificador de áudio HE-AAC v2 no DSP Blackfin BF527 como parte do projeto Rede H.264,cujo objetivo é o desenvolvimento de produtos de interesse nacional na área do ISDB-Tb. É apresentado um breve relato da metodologia e dos resultados obtidos.

5.1 A ESCOLHA DO CÓDIGO DE REFERÊNCIA

A etapa inicial para o desenvolvimento deste trabalho foi a realização de um amplo estudo sobre a tecnologia e as aplicações já existentes no mercado. O primeiro código analisado foi o *software* de referência ISO/IEC 14496-5 [27] que pode ser encontrado juntamente com asdemais normas do padrão MPEG-4. Nesse código de referência são apresentados *softwares*de referência para demonstrar e esclarecer várias partes do padrão MPEG-4. Sendo assim, essa referência não trata exclusivamente da codificação de áudio no MPEG-4, mas também de diversos outros processos envolvidos na codificação de objetos audiovisuais.

A única aplicação de código aberto que estava de acordo com o MPEG-4, inclusive com as implementações do SBR e do PS, foi o*Enhanced*aacPlus *general audio codec*(3GPP TS 26.401)[28]desenvolvida pelo*European Telecommunications Standards Institute* (ETSI)como parte do *3rd Generation Partnership Project*(3GPP). Este *software* é desenvolvido com código aberto em linguagem ANSI C.

Outro *software* encontrado foi o MPEG-4 HE-AAC v2 Decoder[29] da Analog Devices que também era aderente às especificações do HE-AAC v2 e que já havia sido embarcado para o DSP em questão para fins comerciais.

Ao final desse estudo sobre o que já existia no mercado, chegou-se à conclusão de que o desenvolvimento desse decodificador poderia ser feito com base emalgum código existentedando ênfase ao processo de otimização de acordo com a estrutura do DSP em questão.Com essa ideia definida, foi feita uma análise minuciosa do decodificador da AD na qual

foi constatado que por se tratar de uma versão demonstrativa do produto original, não disponibilizava a implementação das principais partesdo código, mantendo-as“escondidas” dentro de bibliotecas inacessíveis ao usuário comum. Diante disso, essa opção foi descartada.

Então com uma análise mais profunda sobre o decodificadordo 3GPP foi verificado que esse código apresentava vários fatores que o credenciavam para ser a referência utilizada neste projeto. Dentre esses fatores,é possível citar o fato de haver uma versão escrita em ponto-fixo e por ter algumas técnicas de otimização já empregadas, principalmente no que diz respeito à construção eficiente das estruturas de repetição. A versão escolhida para ser o código de referência foi a 3GPP TS 26.411-900desenvolvido pela ETSI [30].

Este *software* foi desenvolvido para o uso em computadores. Seu funcionamento é baseado na criação de um arquivo executável que, em um segundo momento, é chamado por meio de linhas de comando em um interpretador de comandos. Isso não satisfaz a necessidade de se criar um decodificador em tempo real e demandará diversas modificações.

5.4 DEFINIÇÃO DE UM CENÁRIO DE TESTES

Após a definição de se trabalhar com um código de referência e defini-lo,passa-se então para a fase de desenvolvimento de um cenário de testes para a validaçãoda implementação do decodificador.Para isso, seguindo os principais decodificadores existentes, foi definido um cenário onde fosse realizada a decodificação de um *streaming*não em tempo real, mas que esteja armazenado em um computador.A partir desse esquema, é possível validar o funcionamento do processo de decodificação em si. Neste momento,ainda não há a preocupação com a criação de um sistema que sempre aguarde a chegada de novos quadrosa serem decodificados. Isso representa uma redução da complexidade computacional necessária para alcançar o objeto final deste decodificador em tempo real.

O cenário de testes definido se assemelha com o cenário visto no demonstrativo da AD, mostrado naFigura 5.1, na qual, por meio da interface USB, é feita a transferência do *streaming* de áudio que está armazenado no computador para o DSP onde o mesmo será decodificado.

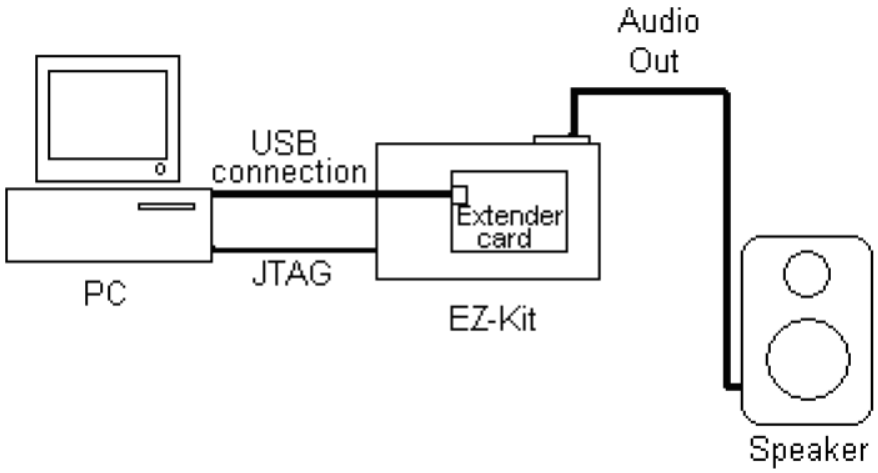


Figura 5.1 - Configuração básica do kit no programa demonstrativo da Analog Devices[29].

5.5 O PROCESSO DE IMPLEMENTAÇÃO

Para o início da utilização do decodificador e para a aplicação das otimizações necessárias, a primeira questão a ser solucionada é a construção um programa principal que seja responsável pelo gerenciamento do processo de leitura e transmissão dos dados, decodificação e distribuição dos dados decodificados.

Para isso, foi desenvolvido um programa que é responsável pelas tarefas acima mencionadas, e também por criar uma interface entre a forma com que os dados são lidos e recebidos e a forma com que o decodificador espera receber os dados. A Figura 5.2 mostra esta necessidade.

Figura 5.2 - Disposição do programa principal entre os módulos existentes.

Mesmo que a saída de um módulo corresponda à entrada do outro módulo se faz necessária a presença de um programa principal que se atente as várias adaptações e correções necessárias para que os dados possam ser decodificados de forma correta.

Dentre os principais ajustes realizados, é possível citar o ajuste no tamanho da estrutura de *buffer* de entrada devido ao fato de cada módulo utilizar um valor e também na forma com que este *buffer* era realimentado.

5.6 A OTIMIZAÇÃO E OS RESULTADOS OBTIDOS

Finalizadas as etapas para portar o código do decodificador de áudio no DSP, tem-se início a etapa de otimização, que corresponde à parte mais importante deste trabalho. Para testar e medir as melhorias obtidas com asotimizações foi utilizado o cenário de testes explicitado anteriormente. Todas as análises realizadas e seus respectivos resultados se referem àdecodificação de um mesmo arquivo de testes. Optou-se por utilizar um único arquivo para facilitar as comparações entre diferentes versões e até mesmo entre decodificadores diferentes. O arquivo escolhido possui as seguintes características:

Nome do Arquivo: `clip2_enc.aac`

Formato do Arquivo: `HE-AAC v2`

Tamanho: `145 KB`

Duração: `26 s`

5.6.1 Avaliação de Desempenho

Por meio da decodificação desse arquivo de teste, é feita a análise para mensurar o desempenho do decodificador. A preocupação com esse desempenho é importante, visto que se deseja obter um decodificador em tempo real e que seja melhorado com a aplicação de técnicas de otimização.

Para esse fim, todas as análises de desempenho feitas durante o processo de otimização baseiam-se em dois itens. O primeiro deles é o tempo necessário para a decodificação do arquivo de testes. Essa medição é definida como o tempo entre o início e o fim da decodificação do arquivo, incluindo todos os tipos de atividades que o programa necessite fazer, por exemplo, acessos à memória e atividades de entrada/saída.Além desta medida de desempenho, também foi levado em consideração a quantidade de ciclos de*clock* necessários para que o programa fosse executado. Os ciclos de*clock* são uma maneira de avaliar o desempenho do *hardware* que é independente da velocidade que o processador está executando.

Dessa forma, utilizando essas duas métricas, é possível ter estimativas relacionadas tanto ao nível de otimização alcançado, quanto à proximidade da execução em tempo real.

5.6.2 Os Primeiros Resultados Obtidos

Ao final do processo de porte do decodificador no DSP, os dados obtidos com a decodificação do arquivo de teste estavam longe de ser ideais. Nessa etapa da implementação, obteve-se valores mais de dez vezes acima do mínimo desejável. Para a decodificação do arquivo de 26 segundos eram necessários 365 segundos e quase 500×10^9 ciclos de*clock*.Isso era algo esperado, visto que o *software* foi desenvolvido para ser executado em outra plataforma com outras características e onde acessos à memória e outras atividades dispunham de muito mais recursos. Foi a partir desse momento que se iniciou o processo de otimização da aplicação.

5.6.3 Aplicação das Otimizações em Relação à Arquitetura do DSP

Para dar início à aplicação das otimizações necessárias para fazer essa implementação atender suas necessidades, a primeira medida e também a que obteve o melhor ganho de desempenho entre todas as otimizações realizadas foi o gerenciamento otimizado de memória através de diversas técnicas de otimização.Os resultados obtidos com o uso dessa e das demais técnicas de otimização foram alcançados utilizando uma relação entre CCLK e SCLK de 5:1, com valores nominais de 600 MHz no CCLK e 120 MHz no SCLK, conforme foi verificado ser os melhores valores para esse tipo de aplicação na seção 4.2.4.

5.6.3.1 Configuração de Memória

Entende-se como gerenciamento otimizado de memória fazer com que o programa utilize de forma eficaz os diversos recursos de memória que o *kit* de desenvolvimento possui. Para isso foram aplicadas as técnicas de otimização em relação à arquitetura do DSP e do *kit* de desenvolvimento do qual ele faz parte.

A primeira medida adotada para otimização foi a habilitação do uso de memória*cache*. Como se trata de um código que é repetido para a decodificação de cada quadro, a manutenção de algumas instruções e dados em memória*cache* representa uma melhoria muito grande.A Tabela 5.1 mostra a comparação da primeira versão, que corresponde aos dados colhidos assim que o decodificador foi embarcado, com os dados colhidos após a utilização da

memória*cache*.

Tabela 5.1 - Comparação entre a versão sem nenhuma otimização e após a habilitação da*cache*.

Versão	Tempo (seg)	Melhoria - Tempo	Ciclos de*Clock*	Melhoria - Ciclos	Observações
Sem Otimização	355	-	492.129.139.854	-	Dados referentes à implementação sem nenhuma otimização
Cache	86	76,44%	121.276.571.627	75,36%	Dados após a habilitação de memória*cache* de dados e instruções

Pela tabela, é possível constatar que realmente se trata de uma otimização com resultados positivos, apresentando em média 75% de melhoria tanto no tempo quanto na quantidade de ciclos necessários para a decodificação.

5.6.3.2 Ajuste de Memória Externa e do*Stack*e*Heap*

No que diz respeito à arquitetura do DSP, foram também aplicadas outras técnicas que também trouxeram melhorias.Uma dessas técnicas aplicadas trata da configuração do*stack* e do*heap*. Essas duas estruturas são utilizadas em tempo de execução pela linguagem C/C++. O*stack* é utilizado pelo compilador para armazenar variáveis locais e endereços de retorno. Já o*heap* é utilizado para armazenamento de dados do tipo alocação dinâmica de memória, por exemplo,por meio da função *malloc*.

Como são duas estruturas que estão ligadas diretamente ao bom funcionamento da aplicação e também serão utilizadas em tempo de execução, é aconselhável que sejam configuradas na memória certa de acordo com a finalidade do código. Para isso,foi feita a configuração do*stack*na memória*scratchpad*, que é um tipo de memória interna do DSP (L1). Já o *heap* por envolver alocações dinâmicas, o que não acontece constantemente nessa aplicação, foi configurado na memória L2. Com esses novos ajustes realizados,foram obtidos novos resultados com ganhos ainda maiores, como pode ser visto na Tabela 5.2.

Tabela 5.2 - Comparação entre a versão atual e após os ajustes do *stack* e *heap*.

Versão	Tempo (seg)	Melhoria - Tempo	Ciclos de *Clock*	Melhoria - Ciclos	Observações
Cache	86	-	121.276.571.627	-	Dados após a habilitação de memória *cache* de
Stack e *Heap*	80	6,98%	110.863.462.454	8,59%	Dados após ajuste do *stack* e *heap*.

Como já era esperado, os novos dados alcançados não representaram um ganho tão grande quanto na otimização anterior. Mas mesmo assim, foram verificadas melhorias de aproximadamente 7% tanto no tempo quanto no número de ciclos.

5.6.3.3 Definição de Seções Prioritárias de Memória

Outra operação realizada para a otimização foi a definição de prioridades entre as funções. Dessa forma, é possível mapear as funções que são executadas com maior frequênciae que requerem maior processamento e colocá-las em seções de memória prioritárias. Essas seções prioritárias são colocadas em regiões onde o processador terá menos trabalho para buscá-las, por exemplo, na memória interna.O contrário também foi realizado colocando funções que não representam gastos computacionais consideráveis em memória externa, aumentando assim o espaço disponível para as funções mais importantes na memória interna.

Como resultado da aplicação dessas duas técnicas citadas, obteve-se novamente algumas melhorias como pode ser visto naTabela 5.3.

Tabela 5.3 - Comparação entre a versão atual e após a definição de seções prioritárias de memória.

Versão	Tempo (seg)	Melhoria - Tempo	Ciclos de *Clock*	Melhoria - Ciclos	Observações
Stack e *Heap*	80	-	110.863.462.454	-	Dados após ajuste do *stack* e *heap*.
Seções de Memória	75	6,08%	101.027.166.432	8,87%	Dados após definições de seções prioritárias de memória

5.6.4 Aplicação das Otimizações em Relação ao Compilador

A otimização do código-fonte faz com que se tenha um melhor uso do compilador. Por essa razão, esta abordagem de otimização é tão importante. Diversas técnicas foram citadas anteriormente e para dar continuidade ao processo de otimização foi feita a aplicação de algumas delas.

5.6.4.1 O uso das Funções Intrínsecas e Funções Inline

As funções intrínsecas representam uma das técnicas de otimização de código que mais trazem benefícios. Isso se dá pelo fato de, ao invés de realizar algumas operações matemáticas dispendiosas de forma convencional,substitui-se por funções que acessem os recursos do processador de forma nativa. Essas funções foram feitas conhecendo o processador, sua estrutura e a forma como realiza as operações. Por esses motivos, a aplicação dessas funções é tão positiva. Outra vantagem é a existência de grupos específicos para DSPs da família Blackfin, o que as torna ainda mais específicas e consequentemente eficazes.

Nesta pesquisa foi feito o uso das funções do grupo *Fractional Value Built-In Functions in C* para substituir funções não otimizadas, pois segundo os dados apresentados na Tabela 4.6esse grupo se demonstrou mais eficaz.

Juntamente com as funções intrínsecas, as funções *inline* também representam significativas melhorias para o desempenho do código por acessarem os recursos especializados do processador ao invés de trabalhar de forma genérica na realização das operações. Os resultados da aplicação dessas técnicas de otimização são mostrados a seguir na Tabela 5.4.

Tabela 5.4 - Comparação entre a versão atual e após o uso de funções intrínsecas e inline.

Versão	Tempo (seg)	Melhoria - Tempo	Ciclos de *Clock*	Melhoria - Ciclos	Observações
Seções de Memória	75	-	101.027.166.432	-	Dados após definições de seções prioritárias de memória
Intrínsecas e inline	39	47,42%	51.084.401.298	49,43%	Dados após o uso de funções intrínsecas e inline

A partir dos resultados expostos na Tabela 4.6 e naTabela 4.7, era possível esperar que o uso destas funções representassemelhorias de aproximadamente 50%, o que as caracterizam como as principais técnicas de otimização de código utilizadas. Mas esses valores, por maiores que tenham sido, ainda estão longe de estarem otimizados como se necessita para a decodificação em tempo real do arquivo de testes.

5.6.4.2 O uso dos Pragmas e Otimização de *Loops*

Para dar continuidade a este processo e alcançar estes requisitos, a próxima etapa de otimização foi através do uso das diretivas de compilação descritas na seção 4.3.5. O uso dessas diretivas voltadas para a otimização do código é muito eficaz e por essa razão representam uma importante etapa do processo de otimização.

A otimização dos laços de otimização (*loops*) é uma etapa fundamental cuja importância justifica-se principalmente pela característica do código que é composto por diversos *loops*. Assim, os ganhos alcançados nessas estruturas vão ser multiplicados pela quantidade de vezes que forem invocados.

A aplicação dessas últimas técnicas representou novos ganhos à execução do decodificador e possibilitou pela primeira vez a sua execução em tempo real, de acordo com os dados apresentados na Tabela 5.5.

Tabela 5.5 - Comparação entre a versão atual e após o uso dos pragmas e otimização dos *loops*.

Versão	Tempo (seg)	Melhoria - Tempo	Ciclos de *Clock*	Melhoria - Ciclos	Observações
Intrínsecas e inline	39	-	51.084.401.298	-	Dados após o uso de funções intrínsecas e inline
Pragmas e *Loop*	26	34,19%	32.746.075.685	35,90%	Dados após o uso dos pragmas e da otimização dos *loops*

Das técnicas aplicadas que geraram os dados apresentados na Tabela 5.5, algumas se destacaram. Dentre as diretivas utilizadas, o pragma `optimize_for_speed` foi um dos grandes responsáveis por alcançar tal nível de otimização. Já das técnicas de otimização de *loop*, o *loop unrolling* foi bastante utilizado e também trouxe um grande ganho de desempenho, conforme já

era esperado pelo visto no capítulo anterior.

5.7Resultados Atuais e Comparações

Com a aplicação de todas as técnicas de otimização acima citadas,foi possível atingir o objetivo inicial de desenvolver um decodificador embarcado em um DSP que tivesse capacidade de decodificar um arquivo de áudio em tempo real. A Tabela 5.6 mostra a evolução alcançada entre a versão inicial e a última versão otimizada.

Tabela 5.6 - Comparação entre a versão inicial e após todas as otimizações.

Versão	Tempo (seg)	Melhoria - Tempo	Ciclos de *Clock*	Melhoria - Ciclos	Observações
Sem Otimização	365	-	492.129.139.854	-	Dados referentes à implementação sem nenhuma otimização
Otimizado	26	92,88%	32.746.075.685	93,33%	Dados após todas as otimizações

Nota-se que o tempo para a decodificação do arquivo correspondeu à duração do arquivo de testes. Mas o fato de terem sido necessários 26 segundos para decodificar o arquivo, não significa necessariamente que o programa esteve ocupado com a decodificação todo este tempo. Em nossa implementação não foi incluída uma função de medida de ciclos ociosos (*idle function*), pois foi reaproveitado um módulo já existente da Analog Devices para as rotinas de entrada e saída de sinais de áudio que não oferece suporte a tal função. Desta forma, mesmo com a continuação do processo de otimização de códigos, a ferramenta *Statistical Profiler* continuaria indicando o mesmo tempo de decodificação do arquivo.

De posse destes dados finais, foi feita uma análise comparativa com o *software* do decodificador da Analog Devices [29] que também tem uma versão embarcada para o mesmo DSP. Os dados referentes à decodificação do mesmo arquivo de testes geraram os dados que são apresentados na Tabela 5.7.

Tabela 5.7 - Comparação entre as o *software* da Analog Devices e o desenvolvido neste trabalho.

Versão	Tempo (seg)	Melhoria – Tempo	Ciclos de *Clock*	Melhoria - Ciclos	Observações
AD	27	-	32.870.170.173	-	Dados referentes à implementação da AD[29]
RT-DSP	26	3,70%	32.746.075.685	0,38%	Dados referentes à implementação apresentada neste trabalho

Como pode ser visto na tabela acima, os dados alcançados pela implementação deste trabalho são superiores ao produto da AD. A implementação do *software* para o ISDB-Tb teve um tempo de um segundo mais rápido e 124 milhões de ciclos de *clock* a menos do que o produto comparado.

Quando se iniciou o processo de otimização, esperava-se alcançar dados tão bons quanto os do programa da AD, que representava uma espécie de referência para análise. No entanto, no decorrer das otimizações,percebeu-se que esses dados poderiam ser até ultrapassados como foi visto ao final do processo de otimização.Isso se deu principalmente,porque a implementação apresentada neste trabalho foi feita exclusivamente para o DSP BF527 segundo suas características.

Mas, por maior que fossem as otimizações realizadas os valores obtidos não seriam muito maiores do que os valores da AD. Isso está relacionado ao fato de, por se tratar de um decodificador em tempo real, o tempo de execução nunca será inferior e não será muito maior do que a duração do arquivo de testes. Como tempo de execução e ciclos de *clock* estão diretamente relacionados, essa variável também não pode estar muito diferente entre os dois decodificadores comparados.

Conclusão

Esta dissertação apresentou o desenvolvimento de um decodificador de áudio de acordo com o padrão HE-AAC v2 do MPEG-4 embarcado em um processador digital de sinais. Além da implementação desse decodificador, foi exposto um estudo de técnicas de otimização para aplicações em sistemas embarcados e a aplicação dessas técnicas no decodificador desenvolvido. Este trabalho foi parte de um projeto em equipe dentro da rede de pesquisa denominada “H.264 SBTVD” em convênio FINEP.

Para a contextualização deste trabalho, os quatro primeiros capítulos foram redigidos de maneira a apresentar ao leitor os subsídios necessários para um melhor entendimento dos procedimentos adotados no desenvolvimento deste trabalho, procurando, contudo, evitar um aprofundamento excessivo dos conceitos, o que poderia tornar a leitura tediosa e cansativa. Para eventuais aprofundamentos, o trabalho conta com uma completa referência, onde o leitor poderá se aprofundar melhor sobre os temas aqui abordados.

Como foi verificado no capítulo dos resultados, todo o processo de desenvolvimento culminou em um protótipo que pode ser utilizado pelo padrão do SBTVD. Esse protótipo desenvolvido, além de atender a todos os requisitos sistêmicos da norma brasileira, também representa uma solução viável para implantação em silício, haja vista a disponibilidade de núcleos ARM com funções de processamento digital de sinais.

Além do desenvolvimento do protótipo final, foi realizadoum apanhado de diversas técnicas de otimização para aplicações embarcadas no ADSP-BF527 e em outros processadores da família Blackfin.

Quanto às perspectivas de trabalhos futuros, existem ainda possíveis aperfeiçoamentos, relacionados à otimização da codificação, como por exemplo, otimização manual em *assembler*, técnica de elevado custo em termos de recursos humanos, que apenas se justifica em sistema de instâncias múltiplas. O decodificadordesenvolvido neste trabalho, assim como o código de referência utilizado, está apto para decodificar apenas arquivos de áudio monofônicos ou estéreos.

Um segundo ponto a ser trabalhado futuramente é a integração do decodificador com um receptor em desenvolvimento pela citada rede de pesquisa. Para isso, algumas alterações

serão necessárias principalmente pelo fato de que neste trabalho, os sinais a serem decodificados estão armazenados em um computador.

Portanto, conclui-se que mesmo havendo melhorias possíveis de serem realizadas, este trabalho alcançou a expectativa inicial de operação em tempo real no DSP alvo. Além disso, este trabalho representa a contribuição deapresentar várias técnicas de otimização de diversos tipos em um único documento, podendo servir como referência para trabalhos futuros de otimização de aplicações embarcadas.

Referências

1. ISO/IEC 11172-3:1993. **Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3: Audio**. ISO/IEC. Geneva. 1993.

2. ISO/IEC 13818-7:2006(E). **Information technology - Generic coding of moving pictures and associated audio information Part 7: Advanced Audio Coding (AAC)**. Geneva, p. 202. 2006.

3. LUO, J.-H. **Design and VLSI Implementation of Low Complexity MDCT-based Psychoacoustic-Model Co-Processor for MPEG-2/4 AAC Encoder**. Taiwan: Tese de Doutorado, National Central University, 2006.

4. MELTZER, S.; MOSER, G. MPEG-4 HE-AAC v2 - audio coding for today's digital media world. **EBU Technical Review**, Janeiro 2006. 1-12.

5. ISO/IEC 14496-3:2005(E). **Information technology - Coding of audio-visual objects - Part 3: Audio**. Geneva, p. 1178. 2005.

6. LEITE, S. B. **Melhoria do codificador de fala G.722.1 através do uso de um modelo perceptual**. Campinas: Dissertação de Mestrado, UNICAMP, 2003.

7. STOLFI, G. **Compressão de Áudio MPEG AAC**, São Paulo, p. 46, 2007. Disponivel em: <http://www.lcs.poli.usp.br/~gstolfi/PPT/Audio_AAC.pps>. Acesso em: 15 Agosto 2010.

8. PEREIRA, F.; EBRAHIMI, T. **The MPEG-4 Book**. Upper Saddle River: Pearson, 2002.

9. BRANDENBURG, K. Wideband Coding - Perceptual Considerations for Speech and Music. In: _____ **Applications of Digital Signal Processing to Audio and Acoustic**. Erlangen: Kluwer Academic Publishers, 2002. p. 39-83.

10. JAYANT, N. S.; NOLL, P. **Digital Coding of Waveforms:** Principles and Applications to Speech and Video. 1ª. ed. Englewood Cliffs: Prentice-Hall, 1984.

11. HERRE, J.; SCHULTZ, D. **Extending the MPEG-4 AAC Codec by Perceptual Noise Substitution**. Audio Engineering Society Convention 104. Amsterdam: [s.n.]. 1998.

12. QUACKENBUSH, S. R.; JOHNSTON, J. D. **Noiseless Coding of Quantized Spectral Components in MPEG-2 Advanced Audio Coding**. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. New Paltz: IEEE. 1997. p. 19-22.

13. HUFFMAN, D. A. A method for the construction of minimum-redundancy codes. **Proceedings of the Institute of Radio Engineers**, Cambridge, Setembro 1952. 1098-1101.

14. GAN, W.-S.; KUO, S. M. **Embedded Signal Processing with the Micro Signal Architecture**. Hoboken: John Wiley & Sons, 2007.

15. ANALOG DEVICES, INC. **ADSP-BF52x Blackfi Processor Hardware Reference**. Norwood, p. 1580. 2010. (Part Number: 82-000525).

16. ANALOG DEVICES, INC. **Data Sheet ADSP-BF52x**. Norwood, p. 88. 2010.

17. ANALOG DEVICES, INC. **VisualDSP++ 5.0 User's Guide**. Norwood, p. 442. 2007. (Part Number: 82-000420-02).

18. KAZTEC ENGINEERING LTD. **System Development and Programming with the ADSP-BF533 Family**. Acton. 2004.

19. DOYLE, D. M. **EE-68 Analog Devices JTAG Emulation Technical Reference**. Analog Devices, Inc. Norwood, p. 20. 2008.

20. OSHANA, R. **Optimizing Compilers and Embedded DSP Software**, 2006. Disponivel em: <http://www.eetimes.com/design/signal-processing-dsp/4016985/Optimizing-Compilers-and-Embedded-DSP-Software>. Acesso em: 09 Setembro 2010.

21. SINGH, K. **EE-271 Using Cache Memory on Blackfin Processors**. Analog Devices, Inc. Norwood, p. 18. 2005.

22. ANALOG DEVICES, INC. **Blackfin Processor Programming Reference**. Norwood, p. 1082. 2008. (Part Number: 82-000556-01).

23. SANGHAI, K. **EE-324 - System Optimization Techniques for Blackfin Processors**. Analog Devices, Inc. Norwood, p. 22. 2007.

24. KERNIGHAN, B. W.; RITCHIE, D. M. **The C Programming Language**. 2ª. ed. New Jersey: Prentice-Hall, 1978.

25. IEEE 754-1985. **IEEE standard for binary floating-point arithmetic**. New York. 1985.

26. ANALOG DEVICES, INC. **VisualDSP++ 5.0 C/C++ Compiler and Library Manual for Blackfin Processor**. Norwood, p. 1248. 2009. (Part Number: 82-000410-03).

27. ISO/IEC 14496-5:2000. **Information technology - Coding of audio-visual objects - Part 5: Reference software**. Geneva. 2000.

28. 3GPP TS 26.401 V9.0.0. **Enhanced aacPlus general audio codec - General description**. ETSI. Sophia Antipolis, p. 13. 2010.

29. ANALOG DEVICES. **MPEG-4 HE-AAC v2 Decoder (with DAB+ and DRM support)**. Disponivel em: <http://www.analog.com/en/embedded-processing-dsp/blackfin/BF_MPEG-4_HE-AAC_V2_DECODER/processors/product.html>. Acesso em: 25 Novembro 2010.

30. 3GPP TS 26.411 V9.0.0. **Enhanced aacPlus general audio codec - Fixed-point ANSI-C code**. ETSI. Sophia Antipolis, p. 23. 2009.