UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ADRIANO RENNER

ARQUITETURA DE UM DECODIFICADOR DE ÁUDIO PARA O SISTEMA BRASILEIRO DE TELEVISÃO DIGITAL E SUA IMPLEMENTAÇÃO EM FPGA

Porto Alegre
2011

ADRIANO RENNER

ARQUITETURA DE UM DECODIFICADOR DE ÁUDIO PARA O SISTEMA BRASILEIRO DE TELEVISÃO DIGITAL E SUA IMPLEMENTAÇÃO EM FPGA

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Engenharia de Computação.

ORIENTADOR: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre

2011

ADRIANO RENNER

ARQUITETURA DE UM DECODIFICADOR DE ÁUDIO PARA O SISTEMA BRASILEIRO DE TELEVISÃO DIGITAL E SUA IMPLEMENTAÇÃO EM FPGA

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Altamiro Amadeu Susin, UFRGS

Doutor pela INPG – Grenoble, França

Banca Examinadora:

Prof. Dr. Sérgio Bampi, UFRGS

Doutor pela Stanford University – Stanford, Estados Unidos

Prof. Dr. Marcelo Soares Lubaszewski, UFRGS

Doutor pela INPG – Grenoble, França

Prof. Dr. Gilson Inácio Wirth, UFRGS

Doutor pela Universität Dortmund – Dortmund, Alemanha

Coordenador do PPGEE: _____

Prof. Dr. Alexandre Sanfelice Bazanella

Porto Alegre, setembro de 2011.

RESUMO

O Sistema Brasileiro de Televisão Digital estabeleceu como padrão de codificação de áudio o algoritmo MPEG-4 *Advanced Audio Coding*, mais precisamente nos perfis *Low Complexity*, *High Efficiency* versão 1 e *High Efficiency* versão 2. O trabalho apresenta um estudo detalhado sobre o padrão, contendo desde alguns conceitos da psicoacústica como o mascaramento até a metodologia de decodificação do *stream* codificado, sempre voltado para o mercado do SBTVD. É proposta uma arquitetura em *hardware* para um decodificador compatível com o padrão MPEG-4 AAC LC. O decodificador é separado em dois grandes blocos mantendo em um deles o banco de filtros, considerado a parte mais custosa em termos de processamento. No bloco restante é realizada a decodificação do espectro, onde ocorre a decodificação dos códigos de Huffman, o segundo ponto crítico do algoritmo em termos de demandas computacionais. Por fim é descrita a implementação da arquitetura proposta em VHDL para prototipação em um FPGA da família Cyclone II da Altera.

Palavras-chave: MPEG-4 AAC, SBTVD, Áudio Digital, AAC LC, Decodificação de áudio, FPGA.

ABSTRACT

MPEG-4 Advanced Audio Coding is the chosen algorithm for the Brazilian Digital Television System (SBTVD), supporting the Low Complexity, High Efficiency version 1 and High Efficiency version 2 profiles. A detailed study of the algorithm is presented, ranging from psychoacoustics concepts like masking to a review of the AAC bitstream decoding process, always keeping in mind the SBTVD. A digital hardware architecture is proposed, in which the algorithm is split in two separate blocks, one of them containing the Filter Bank, considered the most demanding task. The other block is responsible for decoding the coded spectrum, which contains the second most demanding task of the system: the Huffman decoding. In the final part of this work the conversion of the proposed architecture into VHDL modules meant to be prototyped with an Altera Cyclone II FPGA is described.

Keywords: MPEG-4 AAC, SBTVD, Digital Audio, AAC LC, Audio Decoding, FPGA.

ÍNDICE DE ILUSTRAÇÕES

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS

AAC	*Advanced Audio Coding*
ASIC	*Application Specific Integrated Circuit*
DAB	*Digital Audio Broadcast*
DFT	*Discrete Fourier Transform*
DSP	*Digital Signal Processing*
FFT	*Fast Fourier Transform*
FPGA	*Field Programmable Gate Array*
HE	*High Efficiency*
IFFT	*Inverse FFT*
IMDCT	*Inverse MDCT*
ICS	*Individual Channel Stream*
IS	*Intensity Stereo*
ISDB-T	*Integrated Services Digital Broadcasting Terrestrial*
LC	*Low Complexity*
M/S	*Mid/Side Coding*
MDCT	*Modified Discrete Cosine Transform*
MPEG	*Moving Pictures Expert Group*
PNS	*Perceptual Noise Substitution*
PS	*Parametric Stereo*
SBR	*Spectral Band Replication*
SBTVD	Sistema Brasileiro de Televisão Digital
SoC	*System-on-a-Chip*

TNS *Temporal Noise Shaping*

VHDL *VHSIC Hardware Description Language*

VHSIC *Very High Speed Integrated Circuit*

SUMÁRIO

1. INTRODUÇÃO

A compressão de áudio é algo presente no cotidiano de grande parte da população mundial, seja na forma de reprodutores portáteis como o iPod, telefones celulares ou até videogames. O uso de algoritmos para comprimir sinais de áudio é interessante para aplicações que necessitam economizar espaço de armazenamento ou transmitir áudio em canais de dados com banda restrita.

O processo de compressão de um sinal de áudio pode ser com perdas ou sem perdas. Nos métodos sem perdas o sinal reproduzido a partir dos dados codificados é uma réplica exata do sinal original. No caso da compressão com perdas, geralmente o sinal reproduzido é percebido pelo ouvinte como sendo semelhante ou igual ao original, mas matematicamente o sinal é diferente.

1.1. O Sistema Brasileiro de Televisão Digital

A televisão é um dos meios de divulgação de informações e entretenimento mais difundidos no mundo, e no Brasil não é diferente. Em 2009, 95.7% dos lares brasileiros incluídos na Pesquisa Nacional por Amostra de Domicílios possuíam pelo menos um aparelho de televisão, superando outros equipamentos eletrônicos como rádios e computadores (IBGE, 2009).

No território brasileiro predomina o chamado sistema analógico de televisão, com sinal de vídeo obedecendo o padrão PAL-M e o sinal de áudio modulado em frequência, ambos utilizando um mesmo canal do espectro, ocupando 6 MHz de banda. O sistema analógico apresenta algumas restrições técnicas se comparado a capacidade dos equipamentos de televisão e *home theater* disponíveis hoje no mercado, como a baixa resolução de vídeo e áudio estéreo de baixa qualidade.

Os avanços em algoritmos de compressão de vídeo e áudio digital permitem atualmente a transmissão de programas de televisão em alta definição e com áudio multicanal de alta fidelidade usando a mesma banda disponível para um canal transmitido no sistema analógico. O Brasil estabeleceu um padrão próprio de televisão digital chamado Sistema Brasileiro de Televisão Digital (SBTVD) baseado no *Integrated Services Digital Broadcasting Terrestrial* (ISDB-T), desenvolvido no Japão, mas utilizando padrões de compressão de áudio e vídeo digital mais modernos e eficientes. O padrão MPEG-4

Advanced Audio Coding (AAC), estado da arte em compressão de áudio digital, foi o escolhido para o SBTVD por ser capaz de atender as necessidades técnicas para a codificação de um programa com múltiplos canais (5.1 no caso do SBTVD) de alta fidelidade usando uma baixa taxa de bits. Outra vantagem desse algoritmo de codificação é estar livre de pagamento de *royalties* por ser um padrão aberto.

O MPEG-4 AAC é considerado o sucessor do MPEG-1/2 Layer III (MP3), que ficou mundialmente conhecido através da disseminação de conteúdo de áudio (em grande parte ilegal) na internet. O AAC utiliza um banco de filtros com maior resolução em frequência e otimizações em outras partes do processo como a codificação de entropia, sendo capaz de atingir a mesma qualidade do MP3 com taxas de bits mais baixas.

A transmissão do sinal de televisão analógico está prevista para cessar em 2016 no Brasil, quando todas emissoras já deverão ter migrado para a tecnologia digital. Para que isso não torne obsoletos os aparelhos de televisão usados na recepção do sinal analógico, será necessário o uso de um conversor externo, chamado de *set-top box*.

1.2. Objetivo da Dissertação

O objetivo do presente trabalho é desenvolver uma arquitetura de *hardware* para um decodificador de áudio compatível com o padrão SBTVD e executar sua implementação em linguagem VHDL. O objetivo a longo prazo do desenvolvimento de um decodificador em *hardware* é a possibilidade do mesmo ser integrado a um *System on Chip* (SoC) que inclua a decodificação de vídeo e legendas no padrão estabelecido pelo SBTVD. O projeto de um SoC que atenda as necessidades de decodificação do SBTVD e sua produção em massa é uma forma de diminuir o custo dos *set-top boxes*, permitindo que as camadas de mais baixa renda da população também tenham acesso a essa nova tecnologia.

2. CODIFICAÇÃO DE ÁUDIO

Algoritmos de compressão são usados para obter representações digitais compactas de sinais de áudio com o propósito de facilitar a transmissão ou armazenamento. O principal objetivo da codificação é utilizar o mínimo número de bits para a representação do sinal mantendo a fidelidade do áudio reconstruído, de forma que ele seja indistinguível do original para o ouvinte. Existem métodos de codificação sem perdas, que são capazes de reconstruir o sinal original perfeitamente, e métodos com perdas, cujo sinal reconstruído não é idêntico ao original.

Os algoritmos de compressão de áudio com perdas são mais interessantes para aplicações onde não é necessária a reconstrução perfeita do sinal original pois resultam em representações mais compactas se comparados aos algoritmos sem perdas. A maior desvantagem dos algoritmos com perdas é a grande complexidade do processo de codificação, necessária para garantir que as distorções do sinal reconstruído sejam mínimas, idealmente imperceptíveis para os ouvintes.

Entre os algoritmos de compressão com perdas, o maior destaque é para a família dos codificadores perceptuais, que aplicam os resultados de pesquisas sobre a percepção dos sons no sistema auditivo humano para gerar algoritmos capazes de controlar a quantidade máxima de ruído introduzido no processo de codificação. Os padrões MPEG-1 Layer III (MP3) e MPEG-4 AAC, estudado nesse trabalho, são exemplos dessa categoria.

O estudo da percepção dos sons pelo sistema auditivo humano é chamado de psicoacústica. Para o entendimento dos algoritmos perceptuais é fundamental a compreensão de alguns princípios da psicoacústica que serão vistos a seguir, porém, antes disso é necessária a definição de alguns conceitos.

2.1. Sound Pressure Level, Loudness e Just Noticeable Difference

A intensidade de um estímulo acústico pode ser quantificada através da unidade de medida *sound pressure level* (SPL). Ela é representada em decibels (dB) relativa a um nível de referência adotado internacionalmente. A expressão (2.1) denota um valor em dB para uma pressão sonora p relativa ao valor de pressão p_0. A constante p_0 é a pressão mínima necessária para que um estímulo senoidal de 1000 Hz seja percebido pelo sistema auditivo humano. A medida SPL é usada em diversas publicações consultadas, principalmente na caracterização

dos níveis de audição.

$$L_{SPL} = 20\log_{10}\left(\frac{p}{p_0}\right) \text{ (dB SPL)}$$
$$p_0 = 20\,\mu\text{Pa} \qquad (2.1)$$

O SPL fornece uma medida relativa à pressão sonora, mas não indica como esse estímulo é percebido. O nível de intensidade percebido é indicado na literatura pelo termo *loudness*, que é uma quantidade perceptual e, diferente da pressão sonora, não pode ser medida diretamente por instrumentos. A intensidade percebida de um som depende da frequência e da pressão sonora devido a resposta em frequência do sistema auditivo não ser constante (HARTMANN, 1998).

A escala de *loudness* é dada em *phons*, onde por definição, se dois tons são percebidos com a mesma intensidade devem ser representados com a mesma quantidade de *phons*. A figura 1 mostra curvas para alguns valores de *loudness* na faixa de frequências audíveis.

A menor diferença perceptível, ou *just noticeable difference* (JND), caracteriza a menor variação da pressão sonora que pode ser percebida por um ouvinte. Estudos indicam que o JND varia com a intensidade do sinal e a tonalidade (HARTMANN, 1998).

2.2. PSICOACÚSTICA

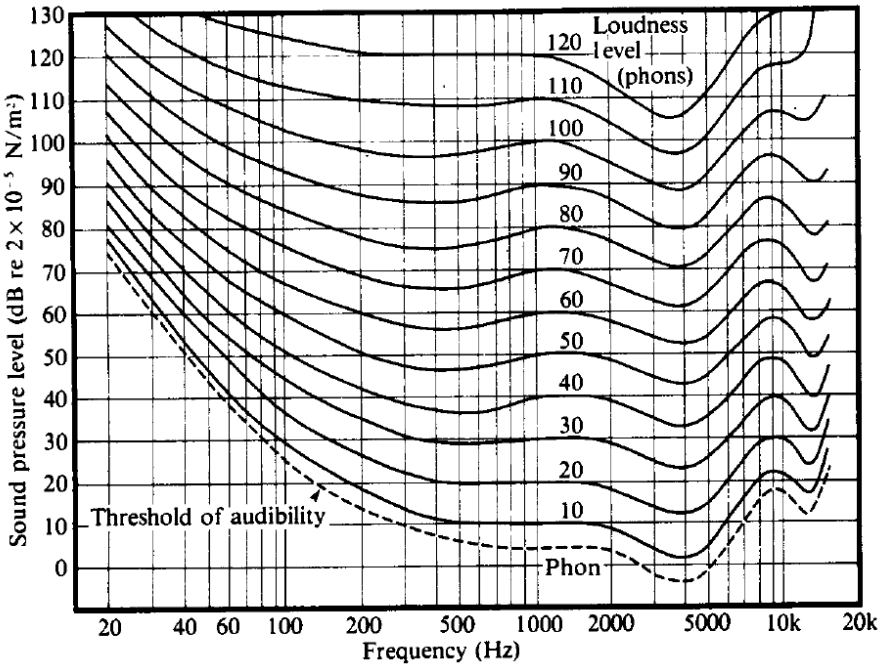


Figura 1: Curvas de igual percepção de intensidade. Fonte:
(HARTMANN, 1998)

Psicoacústica é o nome dado ao estudo da percepção do som pelo sistema auditivo humano, desde a captação da variação de pressão gerada pela onda sonora até a sua conversão em tons. Através da caracterização das capacidades e limitações de percepção de frequências do sistema, algoritmos de codificação são capazes de identificar partes relevantes do sinal de áudio e controlar a sua compressão, minimizando distorções perceptíveis na reprodução resultantes do processo de codificação. Informações irrelevantes são detectadas na análise do sinal incorporando ao codificador alguns princípios psicoacústicos como limite de audição absoluto, análise de frequência em bandas críticas, mascaramento simultâneo, espalhamento do efeito de mascaramento ao longo da membrana basilar e mascaramento temporal.

2.2.1. Sistema auditivo

O sistema auditivo humano é composto pelo ouvido externo, ouvido médio e ouvido interno. O ouvido externo é formado pela porção externa do ouvido e o canal do ouvido, através do qual a onda de pressão gerada por um estímulo acústico chega ao tímpano. O tímpano vibra com a onda e essa vibração é amplificada pelos ossículos do ouvido médio (martelo, bigorna e estribo), sendo transmitida para a cóclea através da janela oval. A cóclea é um canal ósseo em forma de espiral que faz parte do ouvido interno, preenchido por um líquido e onde se encontra a membrana basilar, que se estende por toda espiral.

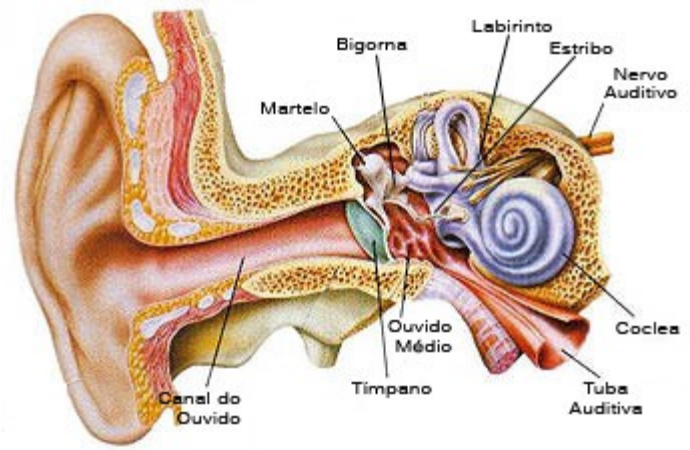


Figura 2: Ilustração do sistema auditivo humano

A movimentação do líquido contido na cóclea causa a excitação da membrana basilar que é convertida em sinais elétricos transmitidos para o cérebro através do nervo auditivo. Ao

longo da membrana basilar é realizada uma transformação de posição para frequência que é interpretada pelo cérebro para a detecção dos diferentes tons em um estímulo sonoro. O estímulo sonoro gera picos de resposta em posições específicas da membrana, sendo a maior resposta aos estímulos de alta frequência obtida na base da cóclea. A frequência de ressonância da membrana basilar diminui conforme a posição observada se aproxima da extremidade interna da espiral, conforme ilustrado na figura 3.

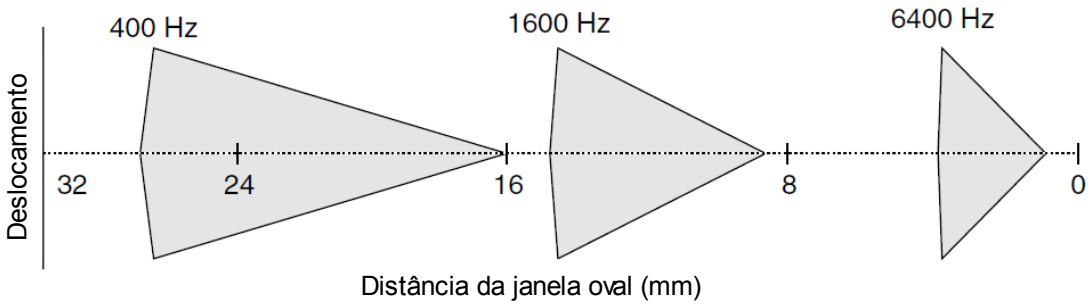


Figura 3: Representação da transformação de frequência para local ocorrida na membrana basilar. Fonte: (SPANIAS, 2007)

2.2.2. Limite absoluto de audição

O limite absoluto de audição caracteriza a energia necessária em um tom puro para que ele possa ser detectado por um ouvinte em um ambiente sem ruído. Esse limite é diferente para cada pessoa, mas pode ser bem aproximado pela equação (3.2) para um ouvinte jovem com audição acurada (SPANIAS, 2007), onde a energia associada a uma frequência f é dada em dB SPL.

$$T_q(f) = 3.64\left(\frac{f}{1000}\right)^{-0.8} - 6.5\,e^{-0.6(f/1000-3.3)^2} + 10^{-3}\left(\frac{f}{1000}\right)^{4} \text{ (dB SPL)} \qquad (3.2)$$

A figura 4 ilustra a função $T_q(f)$ para a faixa de frequências percebidas pelo sistema auditivo. Através da representação gráfica é possível perceber que o sistema apresenta um comportamento não linear e é mais sensível aos sons com frequência próxima de 3 kHz. Devido a essa característica a intensidade percebida para dois sinais de mesma pressão mas de frequências diferentes não é necessariamente a mesma. Como exemplo podemos supor dois

tons no nível de 20 dB SPL, o primeiro a uma frequência de 200 Hz e o segundo a 2 kHz. A intensidade sonora percebida em razão do segundo tom é maior que a percebida decorrente do primeiro, apesar de ambos estarem no nível de 20 dB SPL.

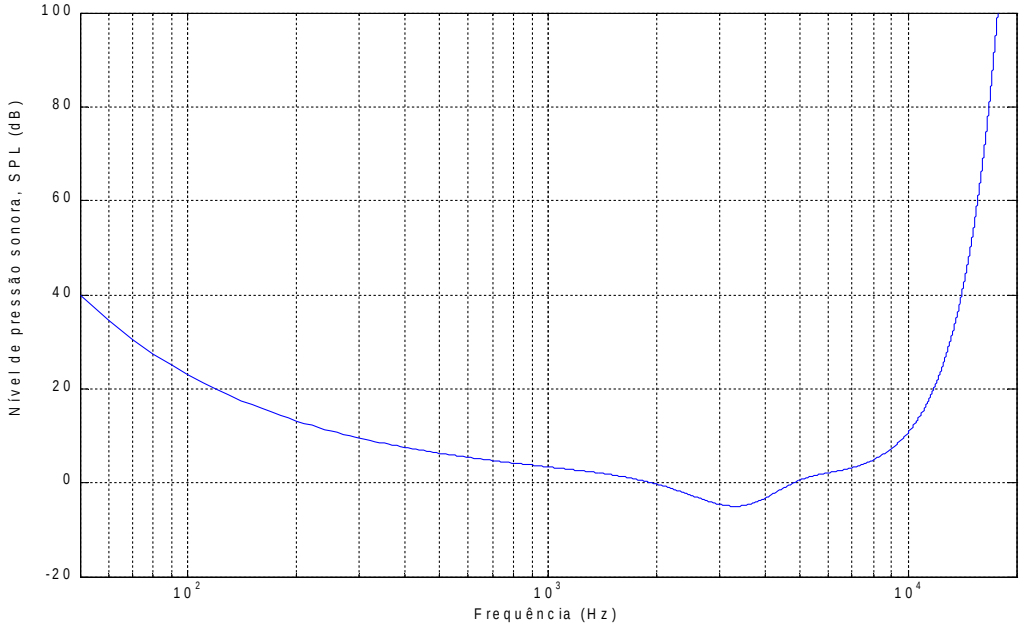


Figura 4: Limite de audição em ambiente sem ruído

2.2.3. Bandas Críticas

A partir da perspectiva de processamento de sinais, o comportamento da membrana basilar é análogo a um banco de filtros passa faixa sobrepostos. A magnitude da resposta dos filtros é não linear, dependendo do nível do sinal, e assimétrica. A largura da faixa de frequência dos filtros não é uniforme e cresce com o aumento da frequência central, diminuindo a resolução espectral (SPANIAS, 2007). Cada canal desse banco de filtros é chamado de filtro auditivo.

Uma banda crítica corresponde a uma distância constante na membrana basilar e à largura de banda dentro da qual a intensidade dos sinais é somada para determinar se a combinação dos sinais excede o limite de mascaramento (BRANDENBURG, 1996).

Um dos métodos experimentais descrito na literatura para determinar a largura de faixa dos filtros auditivos envolve a detecção de um ruído de banda estreita situado no

domínio frequência entre dois tons mascarantes. A distância em frequência entre os tons é variada ao longo do experimento enquanto o ruído é mantido centrado na frequência original. O limite de detecção do ruído se mantém constante enquanto a distância dos tons mascarantes é menor que uma banda crítica, caindo rapidamente quando a distância ultrapassa esse limite.

Para um ouvinte jovem com a audição dentro dos limites considerados normais, tradicionalmente as bandas críticas podem ser aproximadas pela equação (3.3) (SPANIAS, 2007) (MOORE, 1996). Nesse modelo a largura das bandas críticas se mantém quase constante em aproximadamente 100 Hz para frequências centrais de até 500 Hz. Acima disso a banda é aproximadamente 20% da frequência central.

$$BW_c(f) = 25 + 75\left[1 + 1.4\left(\frac{f}{1000}\right)^2\right]^{0.69} \quad \text{(Hz)} \tag{3.3}$$

A escala gerada através do mapeamento das frequências para número das bandas críticas é chamada Bark (3.4). A distância de um Bark corresponde a uma banda crítica, sendo os valores inteiros da escala associados à frequência da extremidade inferior de uma banda crítica.

$$Z_b(f) = 13\arctan(0.00076\,f) + 3.5\arctan\left[\left(\frac{f}{7500}\right)^2\right] \quad \text{(Bark)} \tag{3.4}$$

Um modelo alternativo de bandas críticas, baseado em dados experimentais obtidos para a caracterização dos filtros auditivos mostra um comportamento diferente para as faixas abaixo de 500 Hz, onde a largura da banda dos filtros se torna menor que 100 Hz (MOORE, 1996).

2.2.4. Mascaramento

O mascaramento se refere ao efeito que ocorre quando um estímulo sonoro se torna inaudível devido a presença de outro estímulo. Na literatura são citados dois efeitos diferentes de mascaramento: o mascaramento simultâneo e o mascaramento não simultâneo ou temporal. Em um conjunto de amostras de um sinal de áudio pertencentes a um intervalo de tempo finito geralmente existem diversos tons e ruídos mascarantes e o resultado do mascaramento

de cada um deve ser combinado para gerar um limite global de mascaramento. Qualquer sinal abaixo do nível global de mascaramento é considerado inaudível.

O mascaramento simultâneo ocorre quando dois ou mais estímulos são recebidos durante o mesmo intervalo de tempo pelo sistema auditivo. No domínio frequência, a amplitude e forma espectral do sinal mascarante e do sinal mascarado são as características determinantes do mascaramento. Um ruído de banda estreita, por exemplo, exibe uma capacidade mascarante muito maior ao mascarar um tom puro do que um tom puro mascarando um ruído de banda estreita. No domínio tempo, relações de fase entre os estímulos também podem afetar o resultado do mascaramento.

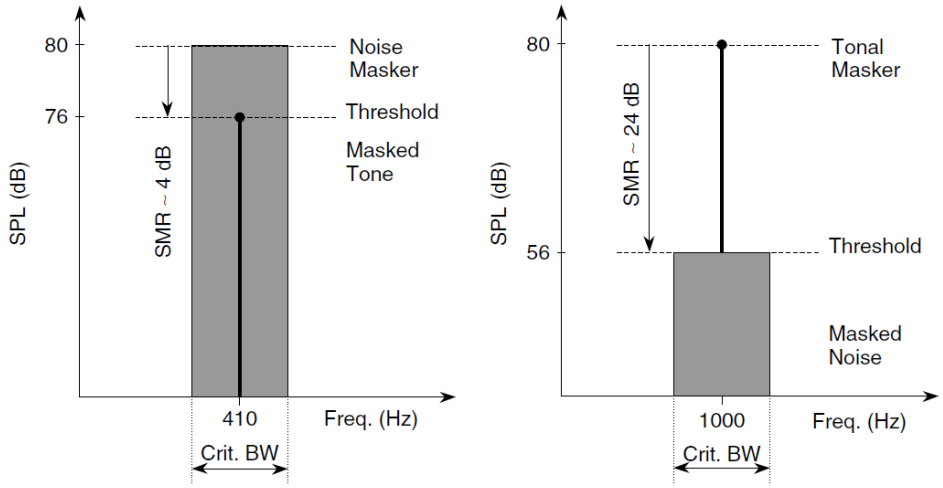


Figura 5: Exemplo ilustrando diferença entre ruído mascarando tom e tom mascarando ruído. Fonte: (SPANIAS, 2007)

Examinando o funcionamento do sistema auditivo é possível entender o efeito de mascaramento simultâneo. Ele se deve à vibração gerada em uma determinada região da membrana basilar por um sinal, que pode ser um ruído ou um tom, denominado mascarante. Essa excitação se espalha pela membrana tornando as regiões próximas menos sensíveis, o que causa o mascaramento de sinais que gerem respostas mais fracas.

O efeito do mascaramento simultâneo não está restrito à banda crítica que contém o sinal mascarante, mesmo que ele seja um tom puro, e o espalhamento entre as bandas críticas deve ser considerado ao determinar o nível de mascaramento global.

O mascaramento causado por um tom puro ou um ruído de banda estreita é caracterizado por uma função de espalhamento. A função de espalhamento define o grau de

mascaramento em frequências diferentes da frequência central do sinal mascarante. Para sinais complexos cada componente do sinal tem a sua própria função de espalhamento, e o efeito combinado dos componentes é uma sobreposição não linear das funções de espalhamento dos componentes. O decaimento do efeito de mascaramento varia com a intensidade do sinal mascarante e a sua tonalidade, mas ele sempre é menos acentuado na direção das frequências acima da frequência central do sinal mascarante, como é ilustrado na figura 6.

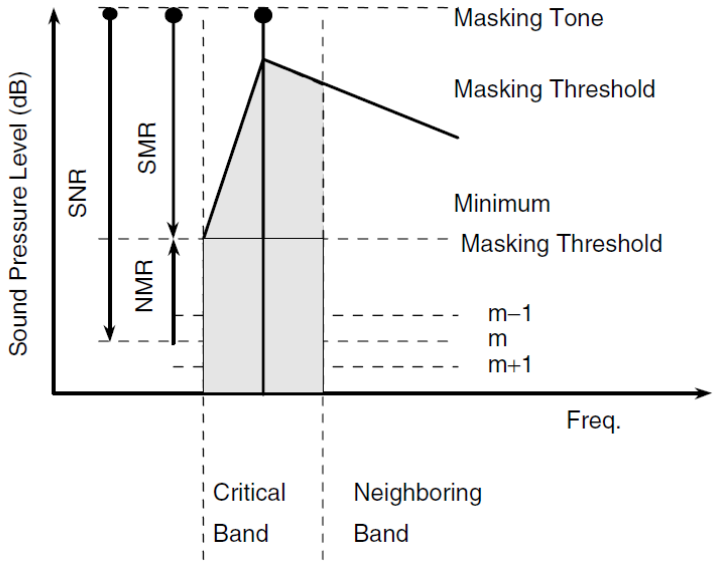


Figura 6: Diagrama de espalhamento do mascaramento gerado por um tom. Fonte: (SPANIAS, 2007)

Os efeitos de mascaramento não simultâneo aparecem antes e depois da ocorrência do estímulo mascarante. A alteração no limite de mascaramento antes do início do estímulo mascarante é chamada de pré mascaramento, enquanto o decaimento do efeito após o fim do estímulo é chamado pós mascaramento. O efeito de pós mascaramento pode se estender por 100 a 300 ms, dependendo da intensidade do sinal mascarante e do tempo de duração do mesmo. O efeito de pré mascaramento, apesar de ter sido submetido a diversos estudos, não é totalmente compreendido (MOORE, 1996) (SPANIAS, 2007). Conforme a figura 7, o efeito de pós mascaramento é muito mais acentuado que o pré mascaramento.

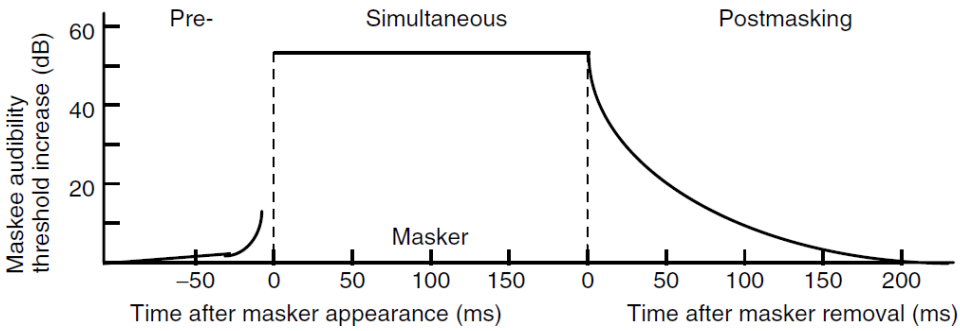


Figura 7: Efeitos de pré e pós mascaramento. Fonte: (SPANIAS, 2007)

2.3. Codificação Perceptual

Os sistemas de compressão de áudio mais eficientes atualmente são os chamados codificadores perceptuais. Sua característica principal é a exploração dos efeitos de mascaramento no sistema auditivo humano para reduzir a percepção de ruídos de quantização (BRANDENBURG, 1996). O principal ponto na codificação perceptual é controlar a quantidade de ruído que pode ser acrescentada ao sinal sem gerar artefatos audíveis. Esse controle é feito usando os efeitos descritos pelos princípios psicoacústicos em um modelo perceptual.

A estrutura básica de um codificador perceptual consiste de um banco de filtros, um módulo de processamento psicoacústico, um bloco de quantização e codificação e um formatador do *bitstream* de saída, conforme a figura 8. O banco de filtros decompõe blocos de amostras do sinal de áudio digital no domínio tempo em seus componentes espectrais. Juntamente com o banco de filtros correspondente no decodificador ele forma um sistema de análise e síntese. Usando o sinal de entrada no domínio tempo ou a saída do banco de filtros de análise, uma estimativa da curva de mascaramento correspondente ao conjunto de amostras de áudio em processamento é calculada no bloco de processamento psicoacústico usando as regras conhecidas da psicoacústica. Os componentes espectrais são quantizados e codificados com o objetivo de reduzir a resolução da codificação (e consequentemente a taxa de bits necessária) enquanto o ruído de quantização é mantido abaixo da curva de mascaramento. Finalmente um formatador de *bitstream* é usado para montar os pacotes de dados contendo os coeficientes espectrais codificados e quantizados e informações adicionais necessárias para a sua decodificação.

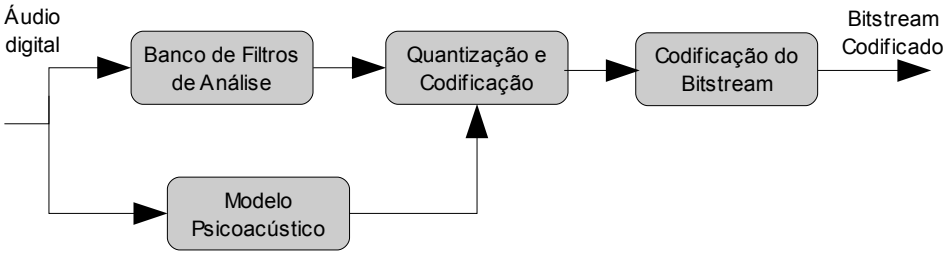


Figura 8: Diagrama de um codificador perceptual básico

No decodificador os dados do *bitstream* são extraídos, passam por um bloco de decodificação no qual os dados quantizados são recuperados e pelo processo de quantização inversa na saída do qual temos os coeficientes espectrais do bloco de áudio. Por fim esses coeficientes alimentam o banco de filtros de síntese, onde bloco de áudio no domínio tempo é reconstruído. A figura 9 apresenta o diagrama de um decodificador perceptual genérico.

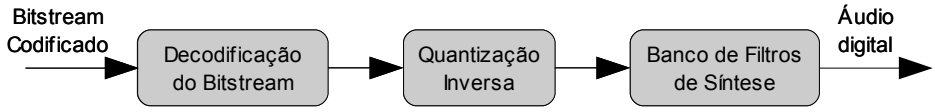


Figura 9: Diagrama de um decodificador perceptual básico

2.3.1. Modelo Psicoacústico

O bloco do modelo psicoacústico é responsável pelo controle do erro introduzido no sinal durante o processo de codificação. Nesse módulo são aplicados os conceitos de mascaramento para gerar uma estimativa do limite de ruído de quantização aceitável em cada conjunto de amostras de áudio.

O modelo psicoacústico pode utilizar a própria saída do banco de filtros ou outro bloco de transformada em paralelo para obter o sinal no domínio frequência. Esse sinal é usado para determinar as curvas de mascaramento na janela de tempo sendo processada. O mascaramento depende do sinal de áudio, portanto varia com o tempo e deve ser recalculado para cada novo bloco de amostras de áudio.

No caso mais simples um modelo estático pode ser usado. No caso de um sistema de codificação no domínio frequência a relação sinal/ruído (SNR) mínima necessária para cada banda crítica pode ser derivada das curvas de mascaramento. Modelos mais complexos

tentam estimar a relação sinal/máscara (SMR) mínima em função do tempo ou um nível de ruído permitido para cada banda em uso no codificador (BOSI, 1997).

A qualidade do modelo usado afeta diretamente a qualidade do sinal codificado, pois o mesmo vai determinar o nível de ruído de quantização acrescentado durante a codificação. O modelo perceptual só é necessário no processo de codificação, o que simplifica muito a decodificação do sinal.

A partir dos resultados obtidos no modelo psicoacústico o bloco de quantização e codificação ajusta a alocação de bits para cada coeficiente espectral.

2.3.2. Bancos de Filtros

Os bancos de filtros no codificador e decodificador constituem um sistema básico de análise e síntese. Historicamente, sistemas de codificação perceptual trabalhando no domínio frequência são chamados de codificadores de sub-banda ou codificadores de transformada. Codificadores de sub-banda normalmente usam poucos canais de seleção de frequências, processando amostras consecutivas. Codificadores de transformada usam um grande número de canais de frequência e processamento simultâneo de amostras adjacentes em frequência.

A decomposição do sinal de áudio em seus componentes de frequência tem papel importante na codificação pois, na maior parte dos casos, o espectro do sinal não é uniforme, o que permite um ganho em termos de eficiência de compressão. Quanto menos uniforme o espectro do sinal maior é o ganho na codificação do mesmo. Caso o sinal no domínio tempo fosse comprimido, a eficiência não seria a mesma.

Sinais de áudio podem apresentar intervalos com transientes abruptos e intervalos com maior conteúdo harmônico. Para obter a melhor redução em taxa de bits, os intervalos com maior conteúdo harmônico necessitam de um banco de filtros com alta resolução em frequência pois seu efeito de mascaramento é bem localizado no domínio frequência. As partes com transientes abruptos apresentam um efeito de mascaramento mais localizado no tempo, e a correta estimativa do seu efeito de mascaramento depende do uso de banco de filtros com maior resolução no tempo. Devido a essa característica muitos algoritmos de codificação perceptual adotam técnicas para alterar a resolução do banco de filtros de acordo com o sinal de entrada.

Diversos tipos de bancos de filtros podem ser usados em um codificador perceptual onde cada um tem características próprias (BRANDENBURG, 1996):

- *Quadrature Mirror Filter* (QMF) em cascata: cada estágio divide a saída anterior em duas sub bandas. Foi usado nos primeiros codificadores de áudio. Apresenta como desvantagem uma alta complexidade, longo atraso e introdução de alias que só é cancelado caso não se use um estágio de quantização.
- *Polyphase filter banks*: composto por bancos de filtros igualmente espaçados combinando a flexibilidade dos bancos QMF com baixa complexidade computacional.
- *Modified Discrete Cosine Transform* (MDCT): também conhecido como *modulated lapped transform* (MLT) ou banco de filtros modulado por cosseno. Reconstrução perfeita desde que atenda algumas condições, como o formato da janela. Usado no MPEG-2 AAC e MPEG-4 AAC.
- Banco de filtros híbrido: composto por uma cascata de diferentes tipos de filtros, é usado no ATRAC e MP3.

2.3.3. Quantização e Codificação

A principal redução na taxa de bits em codificadores perceptuais ocorre no bloco de quantização e codificação. Na quantização a alocação de bits para cada componente espectral é feita de acordo com os dados do modelo psicoacústico, onde o objetivo é representar cada componente com o número mínimo de bits mantendo o ruído de quantização abaixo do nível perceptível, o que evita distorções na reprodução.

A parte de codificação reduz a taxa de bits explorando as redundâncias estatísticas do sinal no domínio frequência após a quantização. Esse procedimento é realizado sem adicionar ruído ao sinal. Um dos algoritmos mais usados para essa codificação é o método de códigos Huffman, descrito na seção 2.4.

Quantizadores uniformes ou não uniformes podem ser usados em conjunto com a codificação. Normalmente o tamanho do passo do quantizador é particular para bandas de frequências que lembram as bandas críticas. Isso é para garantir o mascaramento do ruído de quantização. Nesse caso a alocação de bits não é calculada de acordo com o modelo perceptual, mas o tamanho do passo do quantizador é adaptado para cada banda, mantendo o ruído de quantização abaixo do nível dado pelo modelo perceptual.

2.3.4. Formatação do Bitstream

A saída do bloco de quantização e codificação é colocada no *stream* juntamente com dados auxiliares que serão necessários para o processo de reconstrução do áudio no receptor. Nesse bloco os dados são organizados em pacotes que variam de acordo com a definição do algoritmo de codificação, para que possam ser recuperados no decodificador.

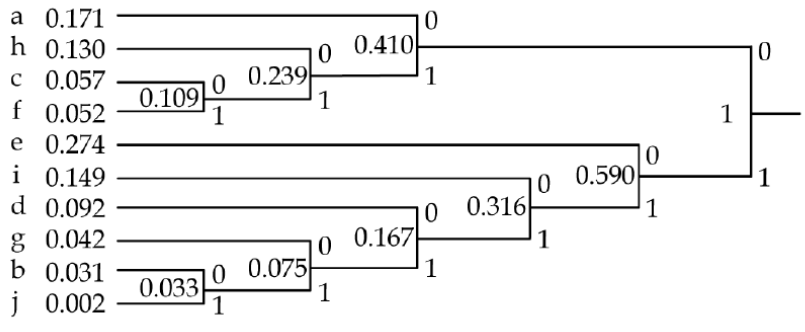
2.4. Decodificação Huffman

O mascaramento e a remoção de redundâncias atuam juntamente para facilitar a codificação de sinais de áudio de alta qualidade usando baixas taxas de bits sem distorções audíveis. Entre os métodos disponíveis para a remoção de redundância do sinal temos a chamada codificação Huffman.

O algoritmo Huffman é um método de codificação de entropia usado para a compressão de dados sem perdas. Os símbolos codificados são representados por palavras binárias de tamanho variável. O tamanho da palavra que irá representar cada símbolo depende da probabilidade de transmissão do mesmo, quanto maior a probabilidade menor o comprimento da palavra. As palavras do código devem ser escolhidas de forma que a representação de um símbolo nunca seja prefixo de outra, por exemplo, se um símbolo é representado pela palavra "01", nenhum outro símbolo pode começar com "01".

A figura 10 mostra como é gerada a tabela de códigos Huffman, onde *X* são os símbolos do alfabeto que pode ser transmitido e *p(X)* é a probabilidade de transmissão de cada símbolo.

X	p(X)
a	0.171
b	0.031
c	0.057
d	0.092
e	0.274
f	0.052
g	0.042
h	0.130
i	0.149
j	0.002



X	Code
a	00
b	111110
c	0110
d	1110
e	10
f	0111
g	11110
h	010
i	1100
j	111111

Figura 10: Exemplo de codificação Huffman

Esse método é usado na maioria dos codificadores de alta resolução em frequência. Se

a distribuição da amplitude espectral é muito irregular, como é o caso de sinais de tom passando por bancos de filtros de alta resolução em frequência, isso pode causar uma grande redução na quantidade de bits necessários para a representação da saída do banco de filtros (BRANDENBURG, 1996).

3. REVISÃO DE LITERATURA

Para o desenvolvimento desse trabalho inicialmente foi estudada a especificação NBR-15602-2 (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2008), que define os parâmetros para os sinais de áudio e o sistema de codificação e decodificação de som para uso no Sistema Brasileiro de Televisão Digital. Segundo os parâmetros especificados para o formato do sinal de áudio temos que a amostragem do sinal deve ser realizada usando 20 ou 16 bits e com frequência de 32 kHz, 44.1 kHz ou 48 kHz. Sobre o sistema de codificação de áudio é especificado que a compressão e os procedimentos de transmissão devem obrigatoriamente ser compatíveis com a ISO/IEC 14496-3 (ISO/IEC, 2005).

A ISO/IEC 14496-3 (ISO/IEC, 2005) especifica o padrão de codificação de áudio MPEG-4. Essa norma é composta por diferentes tipos de codificação de áudio com aplicações vão desde a codificação de voz em tempo real para sistemas de telefonia até a compressão de áudio de alta fidelidade. Para o desenvolvimento de um decodificador de áudio compatível com o padrão SBTVD as seções 1 e 4 da norma, denominadas *Main* e *General Audio coding* respectivamente, foram estudadas.

Na seção 1 da ISO/IEC 14496-3 (ISO/IEC, 2005) são definidos os tipos de objetos de áudio do padrão, os perfis de codificação e os níveis de cada perfil. Esses parâmetros servem para identificar o conjunto de ferramentas usadas para codificar o *stream* de áudio, frequência de amostragem do sinal e quantidade máxima de canais. Os perfis de codificação do SBTVD são restritos ao AAC (ou *Low Complexity* no padrão MPEG-2), *High Efficiency* versão 1 e *High Efficiency* versão 2.

A seção 4 da mesma norma (ISO/IEC, 2005) contém a especificação de algoritmos de compressão tipicamente usados para codificar sinais com conteúdo musical de um ou mais canais. O algoritmo *Advanced Audio Codin*g, usado para a codificação do áudio do SBTVD, tem o seu processo de decodificação completamente especificado nessa seção. Em 4.1.1 são apresentados os diagramas de blocos do processo de codificação e decodificação, e os blocos são brevemente descritos sendo chamados de ferramentas. A sintaxe do *bitstream* AAC, com os campos contidos nos pacotes e tamanho em bits de cada parâmetro, é apresentada na subseção 4.4 através de trechos de pseudo código. A decodificação das estruturas de dados que podem estar presentes no *bitstream* é explicada na seção 4.5.2 da norma, onde são definidos o *raw data block* e os elementos sintáticos do padrão. A seção 4.6 do padrão,

denominada *GA-Tool Descriptions*, contém maiores detalhes sobre cada ferramenta usada no processo de decodificação, incluindo descrições do processo de decodificação. Nessa seção o estudo foi restrito ao conjunto das ferramentas previstas pelos perfis utilizados no SBTVD: *Quantization*, *Scalefactors*, *Noiseless coding*, *Joint coding* (*Mid/Side*, *Intensity stereo* e *Coupling channel*), *Temporal noise shaping*, *Filterbank and block switching*, *Perceptual noise substitution* e *Spectral band replication*.

Em (SPANIAS, 2007) são abordados diversos tópicos relativos à codificação de áudio, desde os princípios psicoacústicos até o funcionamento dos bancos de filtros. Essa fonte de informações foi extremamente importante para o entendimento dos aspectos teóricos por trás do funcionamento do algoritmo AAC. O capítulo 3, sobre quantização e codificação de entropia, possui uma introdução à codificação Huffman, usada na ferramenta *noiseless coding*. O capítulo 5 possui informações essenciais para o entendimento dos princípios psicoacústicos usados na codificação AAC. Nele são apresentados os conceitos de limiar de audição, bandas críticas e mascaramento. O capítulo 6 é dedicado aos bancos de filtros e transformadas usadas em diferentes algoritmos de compressão de áudio e a seção 6.7 trata da transformada usada no AAC, a *Modified Discrete Cosine Transform*. Nas seções 6.9 e 6.10 a distorção chamada de pré-eco é explicada e são descritos métodos usados para evitar esse tipo de efeito, como o chaveamento entre transformadas de tamanhos diferentes e o modelamento do ruído de quantização no domínio tempo. No capítulo 10 são explorados diversos padrões de compressão de áudio, sendo de maior interesse para o trabalho realizado a seção 10.4, que trata dos padrões MPEG.

Diversos artigos contendo informações teóricas sobre codificação perceptual foram estudados para complementar o estudo do processo de compressão. Uma introdução aos conceitos básicos dos codificadores perceptuais é dada em (BRANDENBURG, 1996), esclarecendo mecanismos de audição, a teoria do mascaramento e a remoção de redundâncias. Também é descrito o modelo básico de um codificador perceptual usando um banco de filtros. apresentando um modelo básico dos codificadores pertencentes a essa família de algoritmos de compressão.

Brandenburg esclarece as tecnologias de compressão de áudio utilizadas nos padrões estabelecidos pelo grupo MPEG em (BRANDENBURG, 1999). O foco principal do texto é a descrição dos componentes do codificador MPEG-1/2 *Layer* 3 (MP3) e a sua comparação com o MPEG-2 AAC, apresentando os aperfeiçoamentos implementados no segundo. Por fim

são expostas considerações sobre a qualidade da compressão desses algoritmos, introduzindo os principais tipos de artefatos de áudio, que são distorções e ruído resultantes do processo de compressão.

Wolters (WOLTERS, 2003) e Ehret (EHRET, 2004) tratam como tema principal o módulo *Spectral band replication* (SBR), adicionado ao padrão MPEG-4 para o perfil *High Efficiency* AAC versão 1. Em (WOLTERS, 2003) a tecnologia do SBR e suas implicações aos sistemas MPEG-4 são abordadas, descrevendo os métodos para integração do novo módulo com os codificadores MPEG-4 AAC e aplicações típicas. Em (EHRET, 2004) são apresentados os benefícios do uso do perfil *High Efficiency* AAC versão 1 para gerar *bitstream* com taxas de bits moderadas, na faixa de 80 a 128 kbit/s para sinais estéreo, onde usualmente era escolhido o perfil AAC.

Liu apresenta em (LIU, 2006) um estudo sobre os artefatos de áudio característicos das técnicas de codificação usadas no AAC. Os tipos de artefatos são caracterizados e é feita uma análise dos módulos do codificador que originam cada distorção. A primeira parte do artigo é dedicada aos problemas causados por restrições na taxa de bits da codificação, que pode gerar vales no espectro ou o descarte da faixa de altas frequências, e pelo espalhamento do erro de quantização em um bloco de áudio no domínio tempo, que pode ocasionar o aparecimento do efeito denominado pré-eco. Nas partes seguintes são apresentados os artefatos introduzidos pelos módulos *Temporal noise shaping*, *Spectral band replication* e *Parametric stereo coding*.

Buscando uma melhor compreensão do funcionamento do algoritmo antes de iniciar a implementação do mesmo em VHDL foram estudadas diferentes realizações do decodificador em linguagem C. A primeira foi o chamado *software* de referência, desenvolvido pelo grupo MPEG em conjunto com a especificação do AAC com o intuito de facilitar a interpretação da norma. O FAAD, um decodificador em *software* de código aberto, também foi usado no processo de aprendizagem. Por fim, uma versão do decodificador contendo as ferramentas do perfil AAC implementada por um grupo da UnB foi modificada e utilizada para gerar vetores de teste no processo de validação dos módulos em VHDL.

Diferentes arquiteturas para o banco de filtros do MPEG-2/4 AAC foram estudadas a partir de 3 publicações. A arquitetura proposta em (TSAI, 2002a) implementa a IMDCT através de um algoritmo recursivo modificado para consumir menos ciclos de processamento ao custo de mais recursos de *hardware*, e combina os módulos de janelamento e de

sobreposição de janelas com a IMDCT para otimizar o compartilhamento de *hardware*. A IMDCT é realizada em 524288 ciclos, o janelamento consome 512 ciclos e a sobreposição de janelas consome mais 512 ciclos. O módulo completo usa 6 multiplicadores e 11 somadores.

A arquitetura apresentada em (LI, 2009) propõe um módulo único para o cálculo da MDCT e IMDCT, podendo ser utilizada para a codificação e decodificação de áudio. O algoritmo usado é baseado na decomposição do cálculo em núcleos de FFT de N/8 pontos. O cálculo de um bloco IMDCT longo consome 262146 ciclos.

Du (DU, 2008) introduz uma arquitetura para a realização da IMDCT baseada em um algoritmo rápido usando um núcleo de IFFT de N/4 pontos. Essa arquitetura é dividida em 3 blocos chamados de pré processamento, IFFT e pós processamento. Os blocos são conectados através de áreas de memória e os processos de janelamento e sobreposição de janelas são implementados dentro do bloco de pós processamento para otimizar o algoritmo. A transformada inversa de um bloco de áudio consome 12288 ciclos e são utilizados 4 multiplicadores e 6 somadores. Essa arquitetura foi escolhida como base para o desenvolvimento da IMDCT apresentado nesse trabalho.

Tsai e Yen (TSAI, 2002b) comparam 4 algoritmos para a quantização inversa nos padrões MP3 e MPEG-2/4 AAC. Todos métodos fazem uso de tabelas reduzidas para o cálculo do valor inversamente quantizado (256 valores para os algoritmos 1, 2 e 3 e 128 valores para o algoritmo 4 contra 8192 valores caso todos valores fossem tabelados) realizando interpolações para as posições que não se encontram nas tabelas. O algoritmo 1 apresentado nessa publicação foi escolhido como modelo para o desenvolvimento do módulo de quantização inversa em VHDL devido a sua maior simplicidade de implementação e menor consumo de operações.

Diversas publicações na área da decodificação MPEG-4 AAC para sistemas embarcados existem, e podem ser classificadas em 3 categorias: implementações usando DSPs comerciais, usando microcontroladores em conjunto com alguns módulos em *hardware* para acelerar os pontos críticos do algoritmo, e usando somente *hardware* customizado.

Tratando de implementações de decodificadores em *hardware* compatíveis com o padrão, Zhang e outros (ZHANG, 2007) relatam a arquitetura de um ASIC para *Digital Audio Broadcast* (DAB) no padrão MPEG-1 Layer II e a adaptação incompleta dessa arquitetura para a realização de um decodificador MPEG-4 HE AAC versão 2. A partir dos dados obtidos na implementação original eles estimam que o decodificador MPEG-4 HE AAC versão 2

possa ser implementado usando 40000 portas lógicas e 400 kbits de memória em um ASIC. São feitas considerações sobre a largura máxima das palavras armazenadas em memória, definida em 24 bits para garantir a precisão das amostras de áudio decodificadas, para otimizar o uso da área do ASIC criando um grande bloco de SRAM para compartilhar com todos módulos do decodificador. O mesmo raciocínio é seguido para a determinação da largura da área de ROM usada para os coeficientes necessários na decodificação, determinada como 18 bits. A arquitetura proposta usa somente um multiplicador de 24 x 18 bits compartilhado entre os módulos para otimizar o consumo de área do ASIC.

Tsai e Liu (TSAI, 2009) apresentam uma implementação de um decodificador MPEG-4 AAC LC de baixo consumo suportando dois canais de áudio em um ASIC. Uma análise do algoritmo demonstra que a parte mais dispendiosa em termos de tempo de processamento é o banco de filtros seguido pela combinação de *parser* e decodificador Huffman. A arquitetura proposta para o decodificador particiona o processo em 4 blocos de acordo com o tipo de operações realizadas, agrupando as partes que compartilham o mesmo tipo de operações, o que proporciona uma otimização no uso do *hardware*. O mesmo *hardware* é compartilhado para a decodificação de sinais estéreo. A decodificação Huffman é otimizada em termos de velocidade de decodificação usando uma matriz de operadores *AND* e *OR* e recebendo 21 bits de entrada em um único ciclo. O fluxo do *bitstream* é controlado por um *barrel shifter* capaz de deslocamentos de até 21 bits em um ciclo. A quantização inversa usa uma tabela reduzida de 256 valores e um circuito para realizar a interpolação dos valores restantes apresentando as mesmas características do algoritmo 1 em (TSAI, 2002b). O banco de filtros é implementado através de um algoritmo rápido usando um núcleo de IFFT de N/4 pontos, e é apresentada uma análise das operações para a realização de um *pipeline*. A implementação em ASIC consome 2.45 mW e é capaz de decodificar um sinal de áudio estéreo a 44.1 kHz usando um relógio de 1.3 Mhz.

4. MPEG-4 AAC

O *Moving Picture Experts Group* (MPEG) foi responsável pelo desenvolvimento de diversos padrões de compressão de áudio, entre eles o popular MP3. O padrão de compressão de áudio MPEG-4 é o mais recente deles, contendo técnicas consideradas o estado da arte nesse campo e que se aplicam a qualquer tipo de sistema de áudio que necessite de compressão avançada, síntese, manipulação ou reprodução. Esse padrão apresenta conjuntos de ferramentas de codificação categorizadas em: codificação de fala, codificação de áudio, codificação sem perdas, ferramentas de síntese, ferramentas de composição, ferramentas de escalabilidade, *upstream*, robustez a erros.

As ferramentas de codificação de fala são voltadas para aplicações que façam uso da compressão de voz, como telefonia, ou para sintetizar a fala realizando interfaces texto para voz. O conjunto de ferramentas para codificação de áudio sem perdas serve para sistemas que necessitem que o áudio recuperado seja exatamente igual ao sinal original. As ferramentas de síntese de áudio foram desenvolvidas para síntese de música e outros sons a partir de uma descrição usando taxas de bits extremamente baixas. Para aplicações interativas as ferramentas de composição são ideais, permitindo a combinação de diferentes "objetos de áudio" em um ou mais canais de áudio para gerar uma trilha sonora.

Esse padrão apresenta uma coleção de diferentes ferramentas que podem ser usadas para a codificação do sinal de áudio, dependendo da aplicação alvo. As possíveis utilizações vão desde a compressão de voz usando taxas extremamente baixas até a codificação de áudio para sistemas de televisão digital usando sinais de alta fidelidade em múltiplos canais.

O MPEG-4 *General Audio Coding* (GA) diz respeito a parte da norma que contém o algoritmo para codificação de sinais de alta fidelidade, inclusive o AAC. O MPEG-4 AAC foi baseado no MPEG-2 AAC e é compatível com *streams* codificados no padrão anterior. O padrão AAC é considerado o sucessor do MP3 por apresentar diversas melhorias que permitem atingir o mesmo nível de qualidade na compressão do sinal de áudio usando uma taxa menor de bits. Diferentes perfis de áudio se encontram definidos no padrão e cada um suporta determinados objetos de áudio, que por sua vez tem um conjunto de ferramentas especificadas no padrão.

O MPEG-4 AAC é capaz de codificar até 48 canais e suporta frequências de amostragem de até 96 kHz. Os perfis definidos para o SBTVD e portanto de maior interesse

para o presente trabalho são:

1. AAC: permite a decodificação de objetos de áudio tipo AAC *Low Complexity* (LC). Usa bancos de filtro (2048 ou 256 amostras), formas de janela, quantização e codificação padrão AAC. Tem suporte às ferramentas opcionais *Temporal Noise Shaping* (TNS), *Intensity Stereo* (IS), *Coupling*, *Perceptual Noise Substitution* (PNS) e *Mid/Side Coding* (M/S).
2. *High Efficiency* (HE) AAC versão 1: igual ao perfil AAC com a adição da ferramenta *Spectral Band Replication* (SBR).
3. HE AAC versão 2: igual ao perfil HE AAC versão 1 com a adição da ferramenta *Parametric Stereo* (PS).

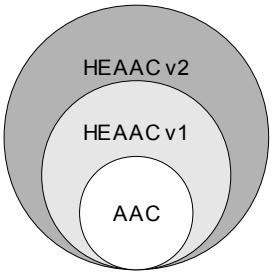


Figura 11: Perfis suportados no SBTVD

4.1. Processo de decodificação do objeto de áudio AAC Low Complexity

O processo de decodificação é separado em blocos, chamados de ferramentas e os dados fluem de forma serial entre elas. O primeiro bloco é chamado de *Bitstream Payload Deformatter*. Nessa ferramenta são extraídas do *bitstream* as informações que irão guiar o processo de decodificação do pacote iniciando pelo tipo de elemento sintático recebido e incluindo dados de configuração essenciais para os outros módulos. Desse bloco se originam os dados com os fatores de escala e espectro quantizado codificados, que são passados para a ferramenta seguinte, chamada de *Noiseless Decoding*.

No bloco *Noiseless Decoding* são decodificados os fatores de escala e os coeficientes espectrais quantizados usando o algoritmo de Huffman, explicado na seção 2.4. Para completar a reconstrução do espectro, os coeficientes espectrais quantizados passam pela ferramenta *Inverse Quantizer*, ou quantizador inverso, sendo o ganho associado aos fatores de escala aplicado aos coeficientes resultantes no bloco *Rescaling*.

O espectro reconstituído passa por um grupo de ferramentas cujo nome é *Spectral Processing*. O uso das ferramentas desse grupo é opcional e cabe ao codificador decidir se elas trarão benefício suficiente para compensar o aumento da complexidade do processo. O *Bitstream Payload Deformatter* recebe através do *bitstream* dados que indicam o uso de cada uma das ferramentas de processamento espectral devendo então ativar os blocos necessários para a decodificação. Esse grupo é constituído pelas ferramentas *Mid/Side* (M/S), *Perceptual Noise Substitution* (PNS), *Intensity Stereo* (IS), *Temporal Noise Shaping* (TNS) e *Dependently Switched Coupling*.

A parte final da reconstrução das amostras de áudio ocorre na ferramenta *Filterbank/Block Switching*, onde a principal operação é a transformação do sinal no domínio frequência para o domínio tempo através de uma IMDCT. A ferramenta *Independently Switched Coupling* atua sobre os dados de saída do banco de filtros, sendo opcional no processo de codificação. Um diagrama com as ferramentas utilizadas para a decodificação do áudio MPEG-4 AAC *Low Complexity* é apresentado na figura 12.

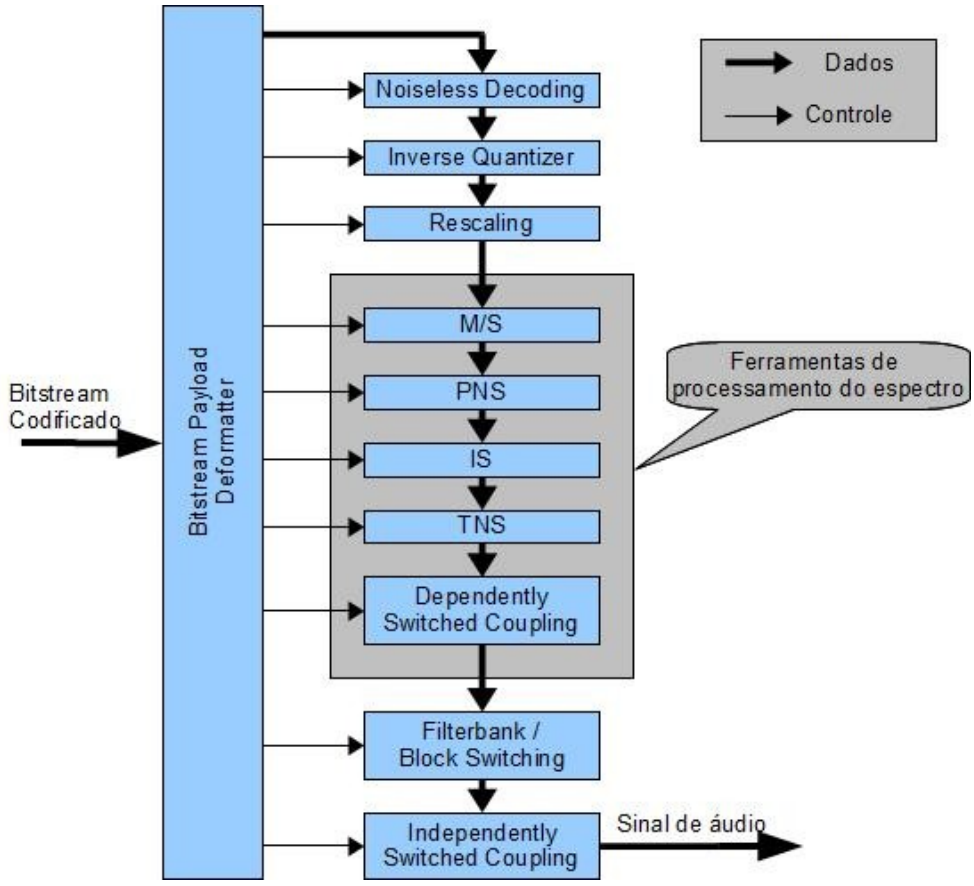


Figura 12: Diagrama do decodificador AAC-LC

4.2. Bitstream Payload Deformatter

O bloco *Bitstream Payload Deformatter* realiza a extração dos dados auxiliares do *bitstream* para uso nas outras ferramentas. O *stream* AAC codificado é organizado em pacotes chamados *Raw Data Blocks*, sendo que cada um possui os dados para reconstruir 1024 amostras de áudio de até 48 canais. Cada pacote é dividido em elementos sintáticos que podem conter dados auxiliares ou dados de áudio.

Tabela 1: Tipos de elementos sintáticos

Identificação do elemento	Código	Nome do elemento
SCE	0x0	*Single Channel Element*
CPE	0x1	*Channel Pair Element*
CCE	0x2	*Coupling Channel Element*
LFE	0x3	*Low Frequency Enhancement Channel*
DSE	0x4	*Data Stream Element*
PCE	0x5	*Program Config Element*
FIL	0x6	*Fill Element*
TERM	0x7	*End Element*

Os tipos de elementos que contém dados de áudio são SCE, CPE, CCE e LFE. O elemento SCE é usado para transportar os dados de áudio de um canal enquanto o CPE possui dados de dois canais de áudio. Dados de áudio do canal de reforço de frequências graves são codificados em elementos LFE. O CCE possui dados de intensidade estéreo no caso de codificação usando um único espectro para dois canais, ou dados para a mixagem de um objeto de áudio na "imagem" estéreo. Os elementos de áudio tem como componente principal o mesmo bloco de dados, chamado *Individual Channel Stream* (ICS), o que facilita o processo de decodificação.

Os elementos SCE e LFE, representados na Figura 13, são estruturalmente iguais. O campo ID identifica o tipo do elemento, nesse caso SCE ou LFE. Além do ID o elemento possui um campo *element instance tag*, que é a identificação do canal, e um bloco ICS. A principal diferença entre o SCE e o LFE é que no ICS do segundo somente os coeficientes espectrais de baixa frequência são diferentes de zero.

O elemento CPE, representado na Figura 14, difere dos anteriores por possuir dados

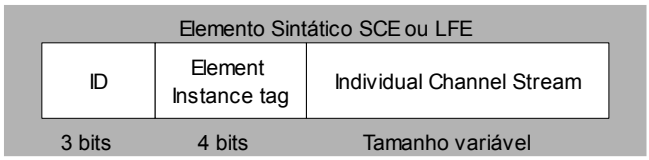


Figura 13: Diagrama de elemento sintático tipo SCE ou LFE

para a reconstrução de dois canais de áudio. Um artifício usado para reduzir o tamanho do pacote na codificação de dois canais é usar um bloco ICS INFO comum para os dois canais. Isso é possível quando ambos usam a mesma configuração de janela de transformada. Quando isso ocorre, o campo *common window* tem valor '1' e os campos ICS INFO e *MS mask present* estão presentes no pacote. Caso *MS mask present* tenha valor '1' o bloco denominado *MS used* também se encontra codificado no pacote. O tamanho desse bloco depende da quantidade de grupos de janelas e do número de bandas de fatores de escala usados na codificação, dados que são transportados no bloco ICS INFO. Os dados *MS mask present* e *MS used* são usados na ferramenta *Mid/Side*. Sempre que *common window* tem valor '0' temos

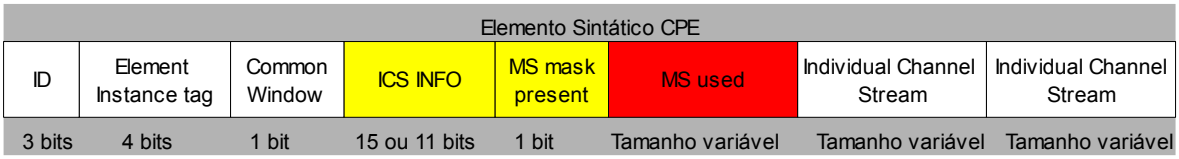


Figura 14: Diagrama de elemento sintático tipo CPE

um ICS INFO dentro de cada bloco ICS, ou seja, um para cada canal codificado.

Os elementos do tipo CCE possuem um bloco ICS e dados específicos para a sua decodificação, que não estão presentes em outros elementos. Esses dados auxiliares trazem as informações relevantes para a decodificação do espectro contido no CCE e para o seu acoplamento com o elemento alvo.

O elemento tipo FIL é usado para transportar dados de extensões do algoritmo, como o *Spectral Band Replication* (SBR) ou para preencher o *bitstream* caso seja necessário manter a taxa de bits constante. Pode transportar até 269 bytes no bloco *Extension Payload*, dependendo dos valores de *Count* e *Esc Count*.

O TERM marca o fim de um *Raw Data Block*, ou seja, delimita os *frames* de áudio codificados. Dados de configuração do *stream* como disposição dos canais de áudio e frequência de amostragem dos canais são codificados dentro de um elemento PCE quando

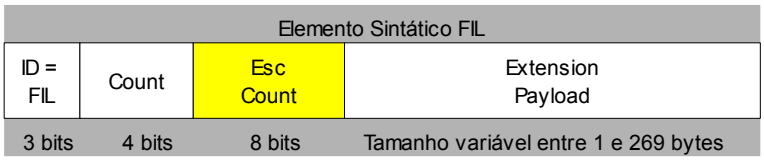


Figura 15: Diagrama de elemento sintático tipo FIL

transportados dentro do *stream* AAC, mas no SBTVD esses dados são enviados na camada de transporte que encapsula os pacotes do AAC. O elemento DSE serve para transportar dados adicionais que não são parte dos dados de áudio e seu uso não está documentado em (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2008).

Todos elementos exceto FIL e TERM possuem um *Element Instance Tag*, que é um campo de identificação do elemento, com 4 bits. Cada elemento que apresenta esse identificador pode aparecer mais de uma vez no mesmo *Raw Data Block* contanto que não repita o valor de identificação, exceto pelo elemento DSE que pode aparecer mais de uma vez no mesmo *Raw Data Block* com o mesmo *Element Instance Tag*. Nos elementos de áudio SCE, CPE, CCE e LFE esse identificador é usado para marcar elementos de um mesmo canal

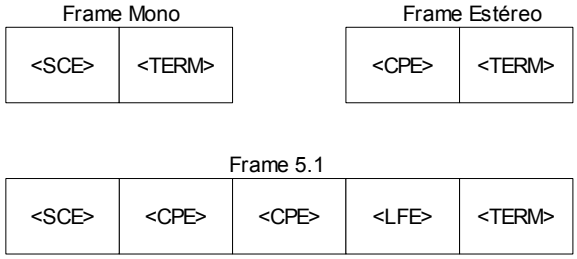


Figura 16: Exemplos de frames recomendados para o SBTVD

de áudio, ou mesmo par no caso do CPE, em uma sequência de *Raw Data Blocks*.

O bloco ICS possui tamanho variável devido às informações de ferramentas opcionais, que podem estar presentes ou não, além do tamanho variável do *bitstream* gerado pela codificação Huffman, que depende da probabilidade dos símbolos sendo codificados. Sempre que o ICS estiver em um elemento SCE ou quando pertencer a um CPE com *common window* '0' o bloco ICS INFO estará presente logo após o parâmetro *global gain*. O bloco identificado na Figura 17 como Dados Opcionais contém os dados para as ferramentas de codificação de pulso e TNS. A codificação de pulso é usada quando alguns coeficientes espectrais excedem o valor máximo de codificação da tabela Huffman usada, forçando a transmissão de pulsos fora do conjunto de dados do bloco *Spectral Data*.

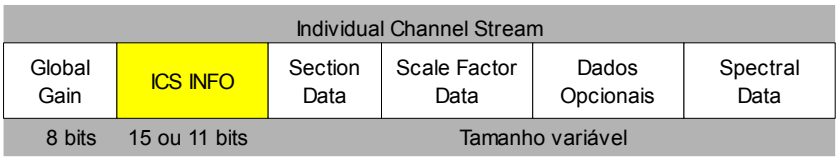


Figura 17: Diagrama simplificado de Individual Channel Stream

O bloco *Section Data* contém a identificação da tabela Huffman usada para codificar cada seção dos coeficientes espectrais e o tamanho de cada seção. Cada seção é composta de uma ou mais bandas de fatores de escala adjacentes que usam a mesma tabela Huffman. Esses dados serão usados no módulo *Noiseless Decoding* para a recuperação dos fatores de escala e dos coeficientes espectrais quantizados. O tamanho desse bloco de dados depende do número de grupos de janelas e da quantidade de seções bloco de áudio.

A decodificação dos blocos *Scale Factor Data* e *Spectral Data* contidos no ICS é realizada pela ferramenta *Noiseless Decoding*.

O bloco ICS INFO carrega os dados de configuração da janela usada na codificação, agrupamento dos fatores de escala no caso de janelas curtas e o número de bandas de fatores de escala em que foram divididas as janelas. O agrupamento de janelas curtas é usado para reduzir a quantidade de dados necessários para codificar os fatores de escala e otimizar a codificação dos coeficientes espectrais através do entrelaçamento dos coeficientes das janelas. Isso é devido a suposição de que janelas agrupadas possuem características estatísticas semelhantes, e com o entrelaçamento os coeficientes correlacionados ficam mais próximos para a codificação de entropia. Esse bloco possui duas possíveis configurações, uma para *window sequence* com valor *Eight Short Sequence* e outra para os valores *Only Long Sequence*, *Long Start Sequence* e *Long Stop Sequence*.

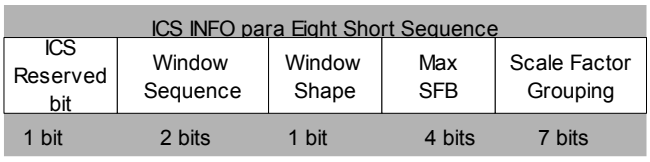


Figura 18: Diagrama de ICS INFO para janelas curtas

No caso do ICS INFO para configuração de janela longa a parte referente aos dados de predição, que está contida no bloco quando *predictor present* é '1', é ignorada pois as ferramentas de predição não são usadas no perfil LC. Isso significa que o campo *predictor present* deve ter sempre valor '0'.

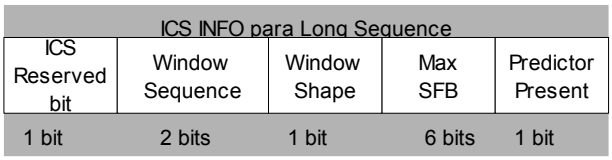


Figura 19: Diagrama de ICS INFO para janelas longas

4.3. Noiseless Decoding

O bloco *Noiseless Decoding* contém as ferramentas de decodificação dos fatores de escala e dos coeficientes do espectro quantizado. A denominação *noiseless*, ou sem ruído em português, é usada pois os valores decodificados por essa ferramenta são idênticos aos que foram codificados, não há inserção de erro no sinal durante o processo de codificação.

No MPEG-4 AAC a codificação Huffman é usada para comprimir os fatores de escala e o espectro quantizado que serão transmitidos no *bitstream*. Para os fatores de escala a norma ISO/IEC 14496-3 (ISO/IEC, 2005) estabelece uma única tabela de códigos Huffman. Como a tabela é conhecida no decodificador a transmissão da tabela não é necessária para a recuperação dos fatores de escala. A quantidade de fatores de escala enviados no *bitstream* depende do tamanho da transformada e da taxa de amostragem do sinal de áudio.

O espectro do sinal é dividido em seções na qual todos coeficientes são codificados com a mesma tabela Huffman. Para a codificação dos coeficientes espectrais existem 11 tabelas na norma. As tabelas são conhecidas no processo de decodificação, mas a identificação da tabela usada deve ser passada como dado auxiliar no *bitstream* para cada seção do espectro. Coeficientes com valores que excedam os representados nas tabelas podem ser transmitidos de duas formas. A primeira é usando a tabela ESC, que permite o recebimento de sinalização de escape seguida dos bits com o valor do coeficiente. A segunda permite que o coeficiente seja substituído por um valor menor para que a codificação Huffman seja mais eficiente. Esse valor é corrigido através de dados auxiliares enviados no *bitstream* com a posição do coeficiente e a diferença no bloco *Pulse Coding*.

O uso de tabelas previamente estabelecidas na norma ISO/IEC diminui a eficiência da codificação Huffman, mas essa perda é compensada pela inexistência da necessidade de transmitir a tabela completa no *bitstream* para o decodificador.

4.3.1. Decodificação dos Fatores de Escala

Os fatores de escala são extraídos da parte do ICS chamada de *Scale Factor Data*. O codificador usa os fatores de escala para controlar de maneira implícita a alocação de bits aplicando ganhos diferentes para cada faixa de frequências (BRANDENBURG, 2000), sempre com o objetivo de manter o erro de quantização abaixo do limiar de audição. Uma faixa de frequências associada a um fator de escala é chamada de *Scale Factor Band*, ou banda de fator de escala, e a largura de cada banda é determinada de forma que elas sejam semelhantes às bandas críticas do aparelho auditivo humano.

A compressão dos fatores de escala é realizada explorando a sua pequena variação entre cada banda, codificando somente a diferença entre os valores adjacentes através da tabela Huffman contida no anexo 4.A do padrão (ISO/IEC, 2005). As diferenças codificadas são inseridas no *bitstream* dentro de *Scale Factor Data*.

Para a decodificação dos fatores de escala é necessário o uso de informações adicionais, que não fazem parte do bloco *Scale Factor Data*. O parâmetro *global gain* é transmitido nos primeiros 8 bits do ICS e serve como valor inicial para o cálculo dos fatores de escala do elemento. A quantidade de fatores de escala codificados é determinada pelo número de bandas de fatores de escala para cada janela da transformada, transmitido no parâmetro *max sfb* dentro do bloco ICS INFO, e do número de grupos de janelas. A identificação da tabela Huffman alocada para uma determinada banda de fatores é determinada através dos dados recebidos em *Section Data*. No caso da tabela usada ser a ZERO_HCB, não existe fator de escala para a banda em questão. Para bandas com as tabelas INTENSITY_HCB ou INTENSITY_HCB2 os fatores de escala são usados para a decodificação da ferramenta *Intensity Stereo*. O valor dos fatores de escala pode variar entre 0 e 255.

O fluxograma da figura 20 foi gerado a partir do pseudo código contido em (ISO/IEC, 2005) e representa o processo de recuperação dos fatores de escala. O processo da decodificação dos fatores de escala inicia lendo o valor de *global gain*, obtido no início do bloco ICS, e guardando em *last_sf*. A decodificação é executada em dois laços, o externo percorre os grupos de janelas começando em 0 e o laço interno percorre as bandas de fatores de escala para cada grupo de janelas, começando em 0 e terminando em *max sfb*. Para cada iteração do laço interno um valor é decodificado usando a tabela de códigos Huffman, o que é representado por *decode_huffman*(), desde que a tabela em uso para os coeficientes espectrais seja válida.

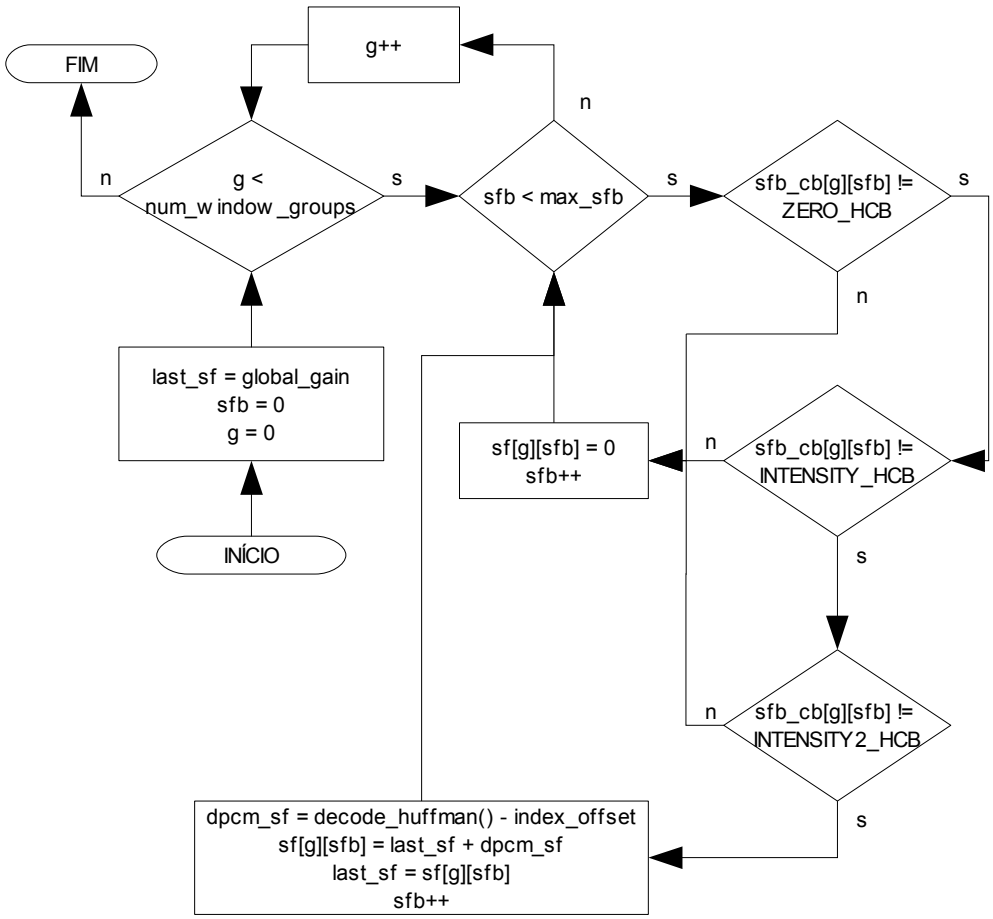


Figura 20: Fluxograma da decodificação dos fatores de escala

4.3.2. Decodificação dos Coeficientes Espectrais

Os coeficientes espectrais quantizados são codificados em grupos de 2 ou 4 valores para aumentar a eficiência da compressão. A decodificação dos coeficientes espectrais quantizados é realizada em partes, a primeira delas a recuperação das palavras codificadas usando uma das tabelas de Huffman definidas na norma (ISO/IEC, 2005). Cada código origina 2 ou 4 coeficientes e os valores podem ser com sinal ou ter o sinal transmitido após a palavra codificada no *bitstream*, dependendo da tabela usada na codificação. Esse agrupamento de coeficientes para a codificação Huffman tem como objetivo explorar a correlação entre os coeficientes adjacentes, aumentando a eficiência da codificação.

O espectro é dividido em bandas de fatores de escala que imitam as bandas críticas do aparelho auditivo, e as bandas são agrupadas em seções. Cada seção é composta pelas bandas de fatores de escala adjacentes que usam a mesma tabela Huffman para a codificação do

espectro. Assim o algoritmo economiza alguns bits na transmissão dos códigos de tabelas Huffman de um *frame*.

4.4. Inverse Quantizer

Os coeficientes espectrais quantizados, obtidos pela decodificação Huffman dos dados de *Spectral Data*, passam pelo processo de quantização inversa representado pela equação (4.1), onde *x_quant* é o valor do coeficiente espectral quantizado, um inteiro com valor absoluto máximo de 8191, e *x_invquant* é o resultado da quantização inversa.

$$x\_invquant = sinal(x\_quant) \cdot |x\_quant|^{\frac{4}{3}} \tag{4.1}$$

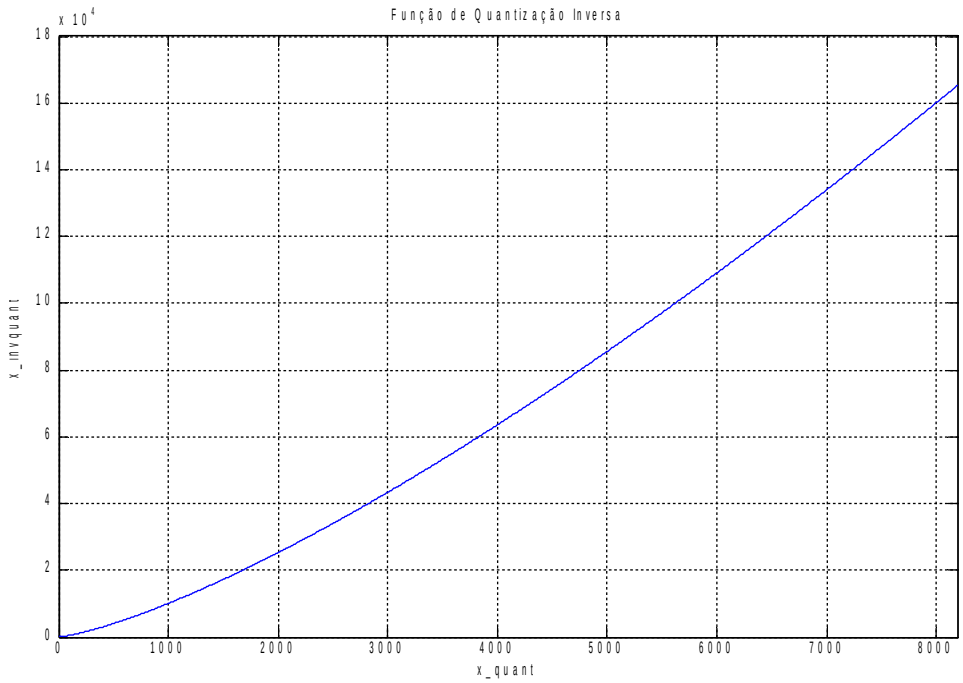


Figura 21: Função de quantização inversa para valores positivos de x_quant

No processo de quantização é introduzido o chamado erro de quantização ao sinal pois *x_quant* só pode assumir valores inteiros. Apesar do erro absoluto introduzido pela função de quantização crescer com o aumento do valor a ser quantizado, o erro relativo ao valor diminui. Essa característica é aproveitada para o modelamento do ruído de quantização através da aplicação de ganhos que são transmitidos na forma dos fatores de escala.

4.5. RESCALING

A aplicação de ganhos através de fatores de escala para determinadas faixas de frequência é o principal método de ajuste do ruído de quantização na codificação AAC (ISO/IEC, 2005).

As faixas de frequência de cada fator de escala são escolhidas de maneira a tentar reproduzir as bandas críticas do sistema auditivo humano. Isso permite que o ganho e consequentemente o ruído de quantização sejam ajustados de maneira independente para cada banda crítica.

Os fatores de escala são usados para calcular o ganho que é aplicado aos coeficientes espectrais contidos na faixa de frequência da banda de fator de escala correspondente. O ganho é calculado pela fórmula (4.2), onde SF_OFFSET é constante com valor 100 e *sf* é o valor do fator de escala.

$$ganho = 2^{0.25(sf - SF\_OFFSET)} \tag{4.2}$$

Os fatores de escala são ajustados durante o processo de codificação para manter o ruído de quantização abaixo do limiar de audição. Quando é detectado que o ruído de quantização se torna audível em uma determinada faixa de frequências, o ganho aplicado a essa banda é aumentado. A consequência disso é o aumento da quantidade de bits alocados para representar os coeficientes da faixa de frequências, o que diminui o erro de quantização.

4.6. FILTERBANK/BLOCK SWITCHING

A reconstrução das amostras de áudio a partir do espectro decodificado ocorre na passagem pelo banco de filtros de síntese. No caso do MPEG-4 AAC LC o banco de filtros é formado por duas *Inverse Modified Discrete Cosine Transform* (IMDCT) de diferentes tamanhos.

O tamanho da janela da transformada é escolhido no processo de codificação de acordo com a análise das características do bloco de sinal sendo comprimido. Quando o áudio possui transientes abruptos a janela de transformada curta é usada devido à sua melhor resolução temporal, que minimiza efeitos de pré-eco. Para sinais quase estacionários, com tons que não se modificam abruptamente, a janela de transformada longa é usada otimizando

a eficiência da compressão. Usando como exemplo uma taxa de amostragem de 48 kHz, a resolução temporal máxima atingida com a transformada curta é de aproximadamente 2,67 ms com uma resolução espectral de 93,75 Hz. A janela longa com a mesma taxa de amostragem apresenta resolução temporal de 21,33 ms e resolução espectral de 23,44 Hz.

A MDCT e sua transformada inversa fazem parte da família de bancos de filtros modulados por cosseno de reconstrução perfeita. O cálculo da MDCT direta, representado por (4.3), é realizado a partir de um bloco de N amostras de entrada usando um avanço de N/2 amostras entre os blocos adjacentes, o que resulta em uma sobreposição de 50%. Para cada bloco de N amostras, N/2 coeficientes são gerados (SPANIAS, 2007).

$$X(k)=\sum_{n=0}^{N-1} x(n)h_k(n),\ \ 0\le k\le\frac{N}{2}-1 \tag{4.3}$$

A reconstrução de um bloco de N/2 amostras necessita entradas de dois blocos consecutivos de coeficientes devido à sobreposição usada na transformada direta (4.4).

$$x(n)=\sum_{k=0}^{N/2-1}\left[X(k)h_k(n)+X^P(k)h_k\left(n+\frac{N}{2}\right)\right] \tag{4.4}$$

O primeiro passo para o cálculo da IMDCT realizado pelo algoritmo do banco de filtros é apresentado pela fórmula (4.5), onde N é o comprimento da janela (2048 ou 256). O comprimento da IMDCT é determinado a partir do parâmetro *window sequence*, que pode assumir um de 4 valores: *Only Long Sequence, Long Start Sequence, Long Stop Sequence* ou *Eight Short Sequence*. Esse parâmetro é extraído do *bitstream* no bloco chamado *Bitstream Payload Deformatter*, descrito na seção 4.2. A IMDCT curta é usada para *Eight Short Sequence*, sendo a versão longa escolhida quando *window sequence* tem qualquer outro valor.

$$\begin{gathered} x(n)=\frac{2}{N}\sum_{k=0}^{\frac{N}{2}-1} spec(k)\cos\left(\frac{2\pi}{N}(n+n_0)\left(k+\frac{1}{2}\right)\right),\ \ 0\le n<N \\ n_0=\frac{1}{2}\left(\frac{N}{2}+1\right) \end{gathered} \tag{4.5}$$

Após a aplicação de (4.5), os dados passam para a aplicação dos coeficientes de janela

e finalmente para a sobreposição com o bloco anterior de áudio. Os coeficientes de janela são determinados por 2 parâmetros, o *window sequence* e o *window shape*. O parâmetro *window shape* é um único bit recebido no *Bitstream Payload Deformatter* que indica a função usada para calcular os coeficientes de janela.

No padrão MPEG-4 AAC para objetos de áudio LC duas funções de janela estão definidas. A primeira delas é uma janela senoidal, representada pelas equações (4.6) e (4.7).

$$w_{\text{sin\_esquerda}}(n)=\sin\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)\right),\quad 0\le n<\frac{N}{2} \tag{4.6}$$

$$w_{\text{sin\_direita}}(n)=\sin\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)\right),\quad \frac{N}{2}\le n<N \tag{4.7}$$

A segunda função de janela apresentada em (ISO/IEC, 2005) é uma *Kaiser-Bessel Derived*, representada pelas equações (4.8), (4.9), (4.10) e (4.11).

$$w_{\text{KBD\_esquerda}}(n)=\sqrt{\frac{\sum_{p=0}^{n} W'(p,\ \alpha)}{\sum_{p=0}^{N/2} W'(p,\ \alpha)}} \tag{4.8}$$

$$w_{\text{KBD\_direita}}(n)=\sqrt{\frac{\sum_{p=0}^{N-n-1} W'(p,\alpha)}{\sum_{p=0}^{N/2} W'(p,\alpha)}} \tag{4.9}$$

$$W'(n,\alpha)=I_0\frac{\left[\pi\alpha\sqrt{1.0-\left(\frac{n-N/4}{N}/4\right)^2}\right]}{I_0[\pi\alpha]},\quad 0\le n\le\frac{N}{2} \tag{4.10}$$

$$I_0(x)=\sum_{k=0}^{\infty}\left[\frac{\left(\frac{x}{2}\right)^k}{k!}\right]^2$$
$$\alpha=\begin{cases}4 \text{ quando } N=2048\\ 6 \text{ quando } N=256\end{cases} \tag{4.11}$$

As funções de janela são dividas em esquerda e direita pois é possível usar uma parte

de cada na mesma janela de áudio, gerando uma janela de transição. Essa combinação pode ocorrer para manter a coerência das janelas na sobreposição, onde as partes que são somadas devem usar a mesma função para evitar a violação dos princípios de reconstrução perfeita (DUHAMEL, 1991). Para que a coerência seja mantida, no AAC a metade esquerda da primeira janela usa o *window shape* recebido no bloco de áudio anterior enquanto a metade direita e as janelas restantes, no caso de haver mais de uma, usam o *window shape* do bloco atual. Assim quando ocorre a mudança de *window shape* o uso da mesma função está garantido na região sobreposta.

A escolha da função de janela é feita de acordo com as características do sinal de entrada visando reduzir o número de componentes de frequência que ultrapassem o limite de mascaramento através da seletividade do filtro. A janela KBD apresenta uma banda de passagem mais larga com maior atenuação na faixa de rejeição, enquanto a janela senoidal tem a faixa de passagem mais seletiva mas menor atenuação na banda de rejeição.

O parâmetro *window sequence* informa como deve ser construída a janela da transformada. Quando possui o valor *Only Long Sequence* a janela obedece (4.12). Essa é a configuração mais simples de janela pois as duas metades usam funções para janelas longas e não é necessário modificar valores dentro da janela.

$$w(n)=\begin{cases} w_{shape\,anterior,2048}(n), & \text{para } 0\le n<1024 \\ w_{shape\,atual,2048}(n), & \text{para } 1024\le n<2048 \end{cases} \tag{4.12}$$

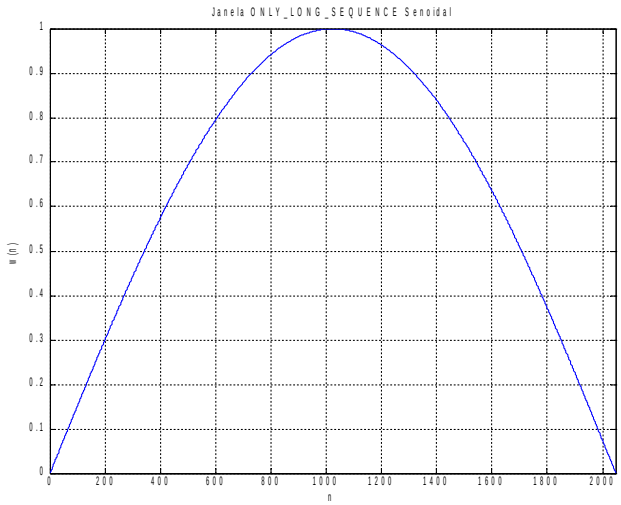


Figura 22: Exemplo de janela *Only Long Sequence* usando função senoidal

As janelas do tipo *Long Start Sequence* são dadas por (4.13). Nesse tipo de bloco assim como no tipo *Long Stop Sequence* é necessário adaptar a janela misturando funções de janela longa e de janela curta. Isso ocorre por serem janelas de transição entre blocos *Only Long Sequence* e *Eight Short Sequence*.

$$w(n)=\begin{cases} w_{shape\ anterior,2048}(n), & \text{para } 0\le n<1024 \\ 1.0, & \text{para } 1024\le n<1472 \\ w_{shape\ atual,256}(n), & \text{para } 1472\le n<1600 \\ 0.0, & \text{para } 1600\le n<2048 \end{cases} \tag{4.13}$$

Blocos do tipo *Eight Short Sequence* possuem 8 janelas curtas e o seu processamento é diferente dos demais pois a sobreposição entre as 8 janelas é realizada junto com a operação de janelamento. As tabelas de coeficientes usadas são de janela curta. Cada janela possui 256

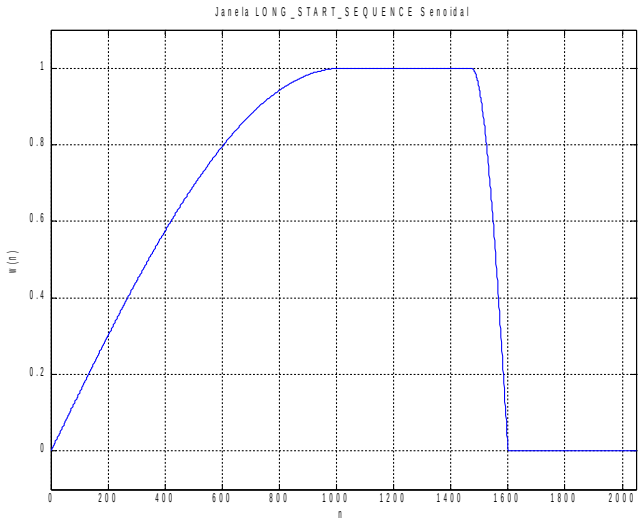


Figura 23: Exemplo de janela *Long Start Sequence* usando função senoidal

amostras que resultam em 2048 amostras no bloco completo, e a sobreposição entre elas é de 50%. O primeiro passo é a multiplicação das amostras pelos coeficientes de janela. Na primeira metade da primeira janela é usada a mesma função de janela do bloco de áudio anterior. Para a segunda metade da primeira janela e todas outras do bloco a função de janela é a correspondente ao *window shape* do bloco atual. O segundo passo é realizar a sobreposição entre as janelas do mesmo bloco. Após calculadas as janelas sobrepostas resultam 1152 amostras. São inseridos 448 zeros antes das 1152 amostras e mais 448 zeros após, completando as 2048 posições do bloco permitindo que a sobreposição entre blocos

completos seja a mesma para todas configurações de janela. As janelas do tipo *Eight Short Sequence* são descritas por (4.14) e (4.15). Em (4.16) são aplicados os coeficientes de janela e é calculada a sobreposição interna das janelas.

$$w_0(n)=\begin{cases} w_{shape\ anterior,256}(n), & \text{para } 0\le n<128 \\ w_{shape\ atual,256}(n), & \text{para } 128\le n<256 \end{cases} \tag{4.14}$$

$$w_{1-(M-1)}(n)=\begin{cases} w_{shape\ anterior,256}(n), & \text{para } 0\le n<128 \\ w_{shape\ atual,256}(n), & \text{para } 128\le n<256 \end{cases} \tag{4.15}$$

$$z_{i,n}=\begin{cases} 0, & \text{para } 0\le n<448 \\ x_{0,n-448}\,w_0(n-448), & \text{para } 448\le n<576 \\ x_{j-1,n-(128j+320)}\,w_{j-1}\left(n-(128\,j+320)\right)+x_{j,n-(128\,j+448)}\,w_j\left(n-(128\,j+448)\right), & \text{para } 1\le j<8,\ 128\,j+448\le n<128\,j+576 \\ x_{7,n-1344}\,w_7(n-1344), & \text{para } 1472\le n<1600 \\ 0, & \text{para } 1600\le n<2048 \end{cases} \tag{4.16}$$

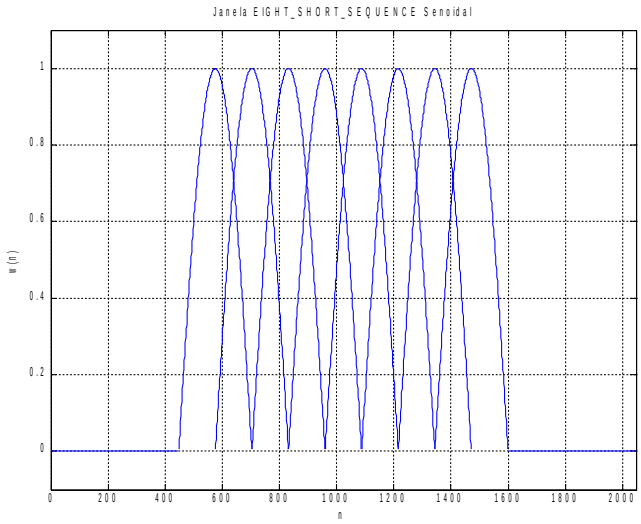


Figura 24: Exemplo de janela *Eight Short Sequence* usando função senoidal

As janelas do tipo *Long Stop Sequence* são representadas por (4.17) e figura 25.

$$w(n)=\begin{cases} 0.0, & \text{para } 0\le n<448 \\ w_{shape\ anterior,256}(n-448), & \text{para } 448\le n<576 \\ 1.0, & \text{para } 576\le n<1024 \\ w_{shape\ atual,2048}(n), & \text{para } 1024\le n<2048 \end{cases} \tag{4.17}$$

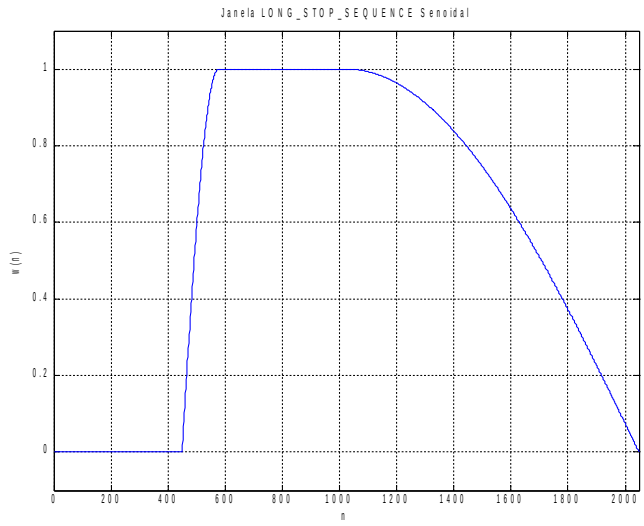


Figura 25: Exemplo de janela *Long Stop Sequence* usando função senoidal

No arranjo *Eight Short Sequence* o *frame* é composto por oito janelas curtas da transformada para possibilitar que diferentes canais possam chavear independentemente entre diferentes resoluções do banco de filtros mantendo o sincronismo no tempo. Para os outros valores de *window sequence* o *frame* é formado somente com uma janela longa. O *frame* do tipo *Long Start Sequence* é usado na transição de *Only Long Sequence* para *Eight Short Sequence* enquanto o *Long Stop Sequence* é usado na transição contrária.

O processo final no banco de filtros é a sobreposição dos *frames*: a primeira metade do *frame* atual é somada à segunda metade do *frame* anterior gerando o áudio reconstruído. A segunda metade do *frame* atual é armazenada para soma com o *frame* seguinte.

4.7. Joint Stereo Coding

A codificação estéreo conjunta reduz ainda mais a quantidade de informações transmitida para o receptor explorando a redundância entre os canais do sinal de áudio estéreo e a irrelevância de alguns componentes (BRANDENBURG, 1996).

Para a grande maioria dos sinais estéreo, a correlação entre os canais esquerdo e direito no domínio tempo é muito pequena. Entretanto, o espectro de potência dos canais muitas vezes apresenta alta correlação. Isso pode ser explicado devido a resolução temporal do banco de filtros usado para obter o espectro do sinal. Quando a diferença de tempo entre os

componentes de sinais correlacionados chegando no microfone esquerdo e direito é menor que a resolução temporal do banco de filtros, temos espectros semelhantes em ambos canais no mesmo período da saída do banco de filtros, em diferentes níveis. Geralmente a redundância estéreo pode ser melhor explorada em sistemas com alta resolução em frequência e consequentemente baixa resolução temporal.

A capacidade do sistema auditivo humano em discriminar a exata posição de origem do áudio diminui com o aumento da frequência. Para altas frequências as informações de impressão espacial são percebidas principalmente através das posições de máxima energia do sinal ao invés dos componentes de fase do sinal.

O MPEG-4 AAC inclui duas técnicas para codificação de sinais estéreo, a codificação *Mid/Side* (M/S), e a codificação *Intensity Stereo* (IS). As duas técnicas podem ser aplicadas em um mesmo *frame* de áudio mas nunca na mesma faixa de frequências (BOSI, 1997).

4.7.1. Mid/Side Stereo

A codificação M/S envolve uma técnica de operação matricial similar a usada em codificadores FM estéreo, junto com a operação inversa apropriada no decodificador. Ao invés de transmitir os sinais esquerdo e direito, a soma normalizada e sinais de diferenças são usados. Eles são denominados canais meio (M) e laterais (S) (BRANDENBURG, 1996).

Sempre que o sinal estiver concentrado no centro da "imagem" estéreo, a codificação M/S pode aumentar a eficiência da compressão (BRANDENBURG, 2000).

Em sistemas com mais de dois canais de áudio, os canais são preferencialmente agrupados de forma que tenham uma posição simétrica em relação ao ouvinte, sendo o primeiro canal do par considerado por definição o esquerdo e sendo o segundo o canal direito.

A ferramenta M/S só pode ser usada quando os dois canais usam a mesma configuração de janela, com o parâmetro *common window* igual a '1'. Nesse caso sempre que o parâmetro *MS used* tiver o valor '1', os coeficientes espectrais da região de frequência correspondente devem ser reconstruídos através da aplicação da matriz M/S inversa (4.18), onde l e r são os canais esquerdo e direito respectivamente, e m e s são os canais *mid* (ou meio) e *side* (ou lateral) respectivamente.

$$\begin{bmatrix} l \\ r \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} m \\ s \end{bmatrix} \tag{4.18}$$

4.7.2. Intensity Stereo

Para cada faixa de coeficientes espectrais de um sinal estéreo transmitida usando a ferramenta IS, somente a informação de um canal é enviada. A informação direcional é transmitida na forma de fatores de escala para canal esquerdo e direito. Assim somente um espectro é transmitido em conjunto com fatores de escala para reconstruir os dois canais (BRANDENBURG, 1996).

O uso da codificação IS é sinalizado no *bitstream* pela escolha somente para o canal direito das pseudo tabelas Huffman INTENSITY_HCB e INTENSITY_HCB2. Quando a tabela usada é a INTENSITY_HCB a codificação IS é em fase, enquanto para a tabela INTENSITY_HCB2 a codificação é fora de fase. Assim como a codificação M/S, o IS só pode ser aplicado em sinais estéreo cujos canais usam a mesma configuração de janela para o mesmo *frame*.

Quando a codificação IS é usada, no lugar dos fatores de escala do canal direito são transmitidas informações de decodificação do IS, chamadas de *intensity stereo position* ou posição de intensidade estéreo. Esses dados relacionam o canal esquerdo com o canal direito para que o segundo possa ser reconstruído a partir do primeiro.

```
p = 0;
for (g = 0; g < num_window_groups; g++) {
    /* Decode intensity positions for this group */
    for (sfb = 0; sfb < max_sfb; sfb++)
        if (is_intensity(g,sfb))
            is_position[g][sfb] = p += dpcm_is_position[g][sfb];
    /* Do intensity stereo decoding */
    for (b = 0; b < window_group_length[g]; b++) {
        for (sfb = 0; sfb < max_sfb; sfb++) {
            if (is_intensity(g,sfb)) {
                scale = is_intensity(g,sfb) * invert_intensity(g,sfb) *
                            0.5^(0.25*is_position[g][sfb]);
                /* Scale from left to right channel,
                   do not touch left channel */
                for (i = 0; i < swb_offset[sfb+1]-swb_offset[sfb]; i++)
                    r_spec[g][b][sfb][i] = scale * l_spec[g][b][sfb][i];
            }
        }
    }
}
```

O pseudo código obtido de (ISO/IEC, 2005) representa o algoritmo de decodificação IS. As funções definidas por *is_intensity* e *invert_intensity* são usadas para o cálculo do ganho a ser aplicado em coeficientes espectrais do canal esquerdo para reconstrução do espectro do canal direito. O vetor *dpcm_is_position* contém os valores decodificados do *stream* no lugar dos fatores de escala.

```
function is_intensity(group,sfb) {
    +1 for window groups / scalefactor bands with right channel
        codebook sfb_cb[group][sfb] == INTENSITY_HCB
    -1 for window groups / scalefactor bands with right channel
        codebook sfb_cb[group][sfb] == INTENSITY_HCB2
    0 otherwise
}

function invert_intensity(group,sfb) {
    1-2*ms_used[group][sfb]   if (ms_mask_present == 1) && aot != AAC scalable
    +1                        otherwise
}
```

4.8. Perceptual Noise Substitution

O estímulo percebido pelo sistema auditivo ocasionado por sinais predominantemente ruidosos é pouco influenciado pelo formato de onda do sinal desde que o formato do espectro dos sinais em questão seja semelhante (HERRE, 1998). Essa característica permite a codificação de sinais de ruído como um parâmetro de energia para aumentar a eficiência da compressão.

No MPEG-4 AAC a ferramenta PNS é responsável pela aplicação desse método. O codificador identifica faixas de frequência onde o sinal possui características de ruído e substitui a transmissão dos coeficientes do espectro pela energia do espectro contido na faixa. Um sinal de ruído é gerado no receptor obedecendo a energia da faixa transmitida no *bitstream* e adicionado ao sinal de áudio.

No *bitstream* o uso da ferramenta PNS é indicado através da seleção de uma pseudo tabela Huffman: NOISE_HCB. A energia do ruído é enviada no lugar dos fatores de escala da faixa de frequências onde será usado o PNS. Assim como os fatores de escala, a energia do ruído é transmitida usando a codificação Huffman dos valores das diferenças. O valor inicial para a decodificação diferencial é o parâmetro *global gain* do ICS.

O pseudo código retirado de (ISO/IEC, 2005) pode ser usado para esclarecer a operação desse bloco. A função *is_noise* serve para identificar se uma determinada banda de fator de escala (*sfb*) de um grupo de janelas (*group*) está usando a ferramenta PNS. Isso é feito verificando a tabela Huffman usada para a região do espectro determinada é a NOISE_HCB.

Antes de iniciar o laço de decodificação dos valores de energia de ruído, a variável *nrg* é inicializada com o valor de *global gain* ajustado por constantes. A substituição dos coeficientes espectrais por ruído é realizada na mesma banda de fator de escala para todas

```
nrg = global_gain - NOISE_OFFSET - 256;
for (g=0; g<num_window_groups; g++) {
    /* Decode noise energies for this group */
    for (sfb=0; sfb<max_sfb; sfb++) {
        if (is_noise(g,sfb)) {
            nrg += dpcm_noise_nrg[g][sfb];
            noise_nrg[g][sfb] = nrg;
        }
    }
    /* Do perceptual noise substitution decoding */
    for (b=0; b<window_group_length[g]; b++) {
        for (sfb=0; sfb<max_sfb; sfb++) {
            if (is_noise(g,sfb)) {
                size = swb_offset[sfb+1] - swb_offset[sfb];
                /* Generate random vector */
                gen_rand_vector( &spec[g][b][sfb][0], size );
                nrg=0;
                for (i=0; i<size; i++) {
                    nrg+= spec[g][b][sfb][i] * spec[g][b][sfb][i];
                }
                sqrt_nrg = sqrt (nrg);
                scale *= 2.0^(0.25*noise_nrg [g][sfb]) / sqrt_nrg;
                /* scale random vector to desired target energy */
                for (i=0; i<size; i++) {
                    spec[g][b][sfb][i] *= scale;
                }
            }
        }
    }
}
```

janelas pertencentes ao grupo. É gerado um vetor de valores aleatórios com tamanho igual ao número de coeficientes espectrais existentes na banda de fator de escala. Por fim o valor da energia de ruído é usado para calcular um ganho que é aplicado ao vetor de valores aleatórios. Os valores desse vetor são então usados no espectro do canal de áudio.

4.9. Temporal Noise Shaping

Artefatos chamados de pré-eco podem ser causados devido à resolução temporal insuficiente do banco de filtros para tratar um sinal com transientes abruptos no domínio tempo (também chamados de ataques abruptos) dentro de uma única janela de transformada. Em alguns casos o uso de janelas curtas de transformada não é suficiente para evitar o desmascaramento do ruído de quantização devido ao espalhamento desse ruído no domínio tempo (HERRE, 1996).

O *Temporal Noise Shaping* (TNS) é um método usado para controlar a distribuição do ruído de quantização no domínio tempo. O conceito por trás desse método é a dualidade da predição linear no domínio tempo e domínio frequência (HERRE, 1999).

Usando um sistema de predição linear no domínio frequência e codificando somente a

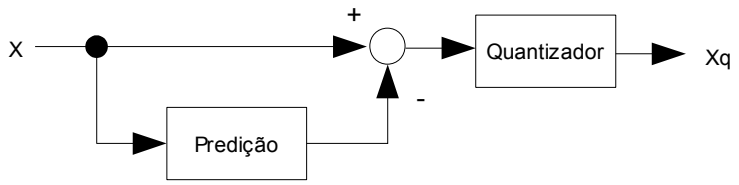


Figura 26: Predição direta de laço aberto

diferença entre o valor atual e o previsto, o erro de quantização se adapta ao formato do sinal original no domínio tempo ao invés de se espalhar pelo *frame*.

O TNS pode ser ativado por banda de fator de escala, sendo o processo realizado em todos coeficientes espectrais contidos nas bandas onde o TNS está em uso. Uma variável de 1 bit no ICS sinaliza o uso do TNS. Os dados necessários para a configuração da ferramenta TNS se encontram no bloco TNS DATA, dentro do ICS.

No processo de decodificação é aplicado um filtro nos coeficientes do espectro das

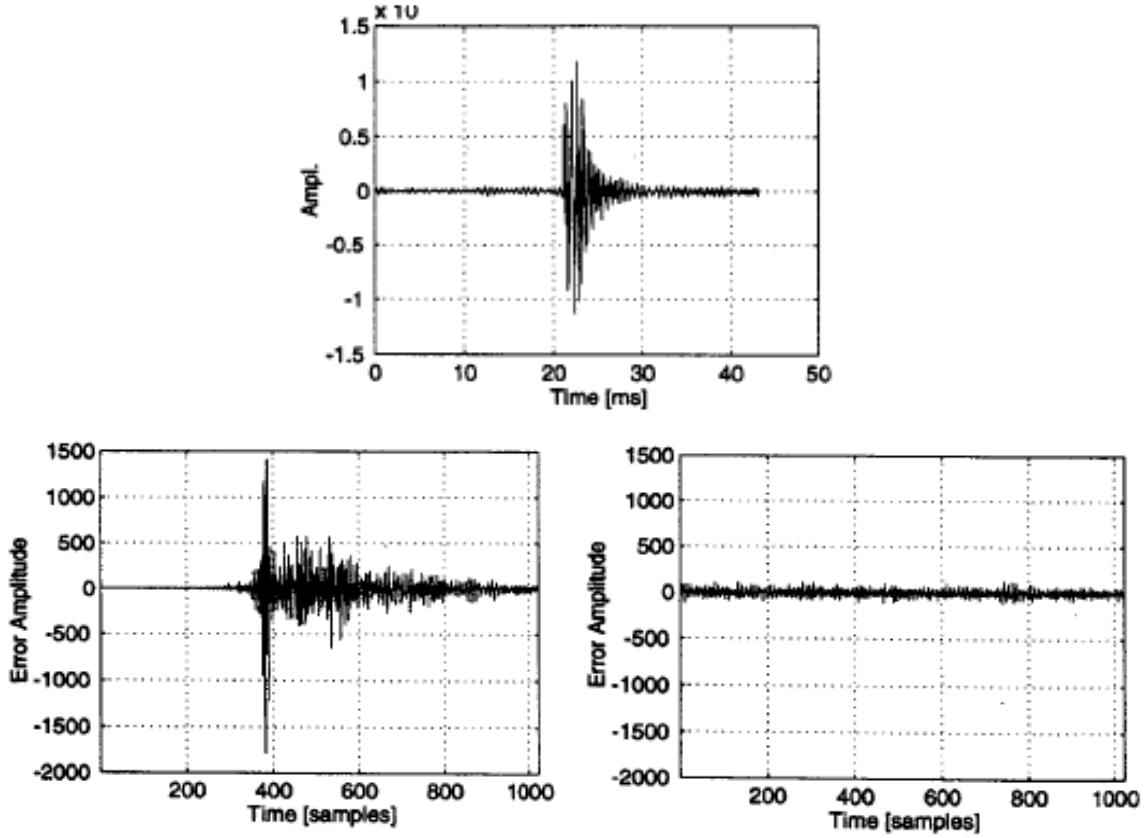


Figura 27: Sinal original acima. Erro de quantização com TNS na esquerda e sem TNS na direita. Fonte: (HERRE, 1999)

regiões determinadas em TNS DATA, realizando o processo inverso da predição. Os coeficientes do filtro são recebidos em TNS DATA também.

4.10. Coupling Channel

A ferramenta *Coupling Channel* está diretamente relacionada ao elemento sintático CCE. Ela tem duas finalidades descritas na literatura: implementar um IS mais geral, podendo compartilhar o espectro contido em outros elementos sintáticos, e realizar a mixagem de canais de áudio adicionais com um sinal estéreo (BOSI, 1997).

O elemento CCE sempre é associado a um SCE ou CPE, através do campo *cc target tag select*. Nesse campo é recebido o *element instance tag* que identifica o elemento alvo do CCE. Além disso, o campo *cc target is cpe* indica se o elemento alvo é um SCE ou CPE.

No diagrama de blocos do decodificador AAC essa ferramenta está dividida em duas partes: *Dependently Switched Coupling* e *Independently Switched Coupling*.

4.10.1. Dependently Switched Coupling

Para que o CCE seja decodificado nesse bloco ele precisa usar a mesma configuração de janela usada pelos canais de áudio associados do elemento alvo. O acoplamento dos canais ocorre no domínio frequência.

Os coeficientes espectrais contidos no CCE devem ser decodificados e armazenados. Esses dados se encontram dentro de um ICS, então essa parte da decodificação é feita como se fosse um SCE. Se elementos de ganho tiverem sido transmitidos no CCE eles são aplicados ao espectro armazenado. O último passo realizado é a adição do espectro do CCE aos coeficientes espectrais contidos no elemento alvo.

4.10.2. Independently Switched Coupling

O elemento CCE é decodificado nesse bloco caso a sua configuração de janela seja diferente da usada nos canais do elemento alvo. Nesse caso o acoplamento dos canais é realizado no domínio tempo.

O bloco ICS contido no CCE deve ser decodificado e o espectro resultante então precisa ser transformado para o domínio tempo, de forma que as amostras resultantes possam ser somadas ao sinal reconstruído dos canais de áudio associados ao elemento alvo.

5. ARQUITETURA PROPOSTA

O trabalho desenvolvido tem como objetivo gerar uma arquitetura de *hardware* para a decodificação básica do AAC LC, atendendo as partes obrigatórias para o decodificador mínimo. Assim, serão estudados métodos para implementar os blocos *Bitstream Payload Deformatter*, *Noiseless Decoder*, *Inverse Quantizer*, *Rescaling* e *Filterbank/Block Switching*.

O decodificador terá suporte a recepção de até dois canais de áudio simultâneos com frequências de amostragem entre 32 kHz e 48 kHz. Nesse primeiro momento as ferramentas espectrais, que são opcionais no processo de codificação, não serão implementadas. Sendo assim, caso alguma seja usada na codificação do *bitstream*, o mesmo não será decodificado corretamente. Toda aritmética necessária para a decodificação será implementada em ponto fixo.

A arquitetura proposta divide o decodificador em dois grandes blocos chamados de Decodificador de Espectro e Banco de Filtros. O primeiro realiza toda a reconstrução dos coeficientes espectrais contidos em um elemento sintático e para isso necessita de acesso direto aos dados do *bitstream.* e realiza a decodificação do espectro do *frame*, enquanto o segundo utiliza os dados do espectro e informações auxiliares para reconstruir as amostras de áudio.

Entre os dois blocos foi colocado um módulo para armazenar os dados de espectro decodificados, permitindo que enquanto o áudio de um *frame* é reconstruído no segundo bloco, a decodificação do próximo possa ser iniciada no primeiro bloco.

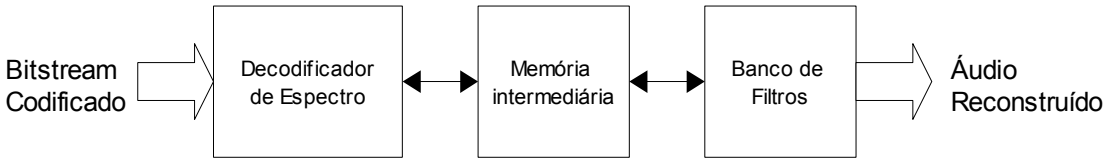


Figura 28: Diagrama de blocos das duas partes do decodificador interconectadas por área de memória

O mesmo banco de filtros será usado para decodificar os dois canais, sendo multiplexado no tempo. Sendo assim, a velocidade da decodificação está limitada por este bloco.

5.1. Decodificação Espectral

O processo de decodificação espectral é realizado em etapas sequenciais, executadas por módulos em *hardware.* Esses módulos são controlados por uma máquina de estados de acordo com o elemento sintático identificado. Todas leituras de dados do *bitstream* ocorrem nesse bloco de operações.

A extração dos dados contidos no *bitstream* foi dividida em diversos módulos para evitar a criação de uma máquina de estados muito complexa. O método adotado para a divisão das etapas foi seguir a segmentação usada na norma (ISO/IEC, 2005) e no código C usado como referência sempre que possível.

Segundo a NBR-15602-2 (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2008) o decodificador deve suportar os elementos SCE, CPE, LFE, FIL e TERM para decodificar o *stream* de áudio do SBTVD. O decodificador proposto não executa a decodificação do elemento LFE visto que em um primeiro momento ele suporta sinais de áudio mono ou estéreo e este elemento somente é usado para transmitir o reforço de baixas frequências. Os dados contidos no elemento FIL são descartados pois são usados para transportar dados de extensões não implementadas, como o *Spectral Band Replication* (SBR).

Não é interessante seguir a arquitetura proposta em (TSAI, 2009) para realizar a decodificação Huffman em 1 ciclo pois a velocidade do decodificador está limitada pela transformada. O objetivo é manter os blocos de decodificação do espectro e de banco de filtros trabalhando em paralelo.

5.1.1. Módulo Bitstream Buffer

O módulo *Bitstream Buffer* foi desenvolvido para controlar o fluxo do *bitstream*, armazenando os dados de entrada e fornecendo para os módulos que necessitam do *bitstream* para a decodificação. A entrada de dados foi dimensionada com 32 bits visando a interface com um processador fornecendo o *bitstream*, de modo que o mesmo não seja sobrecarregado com excessivos pedidos de dados do módulo em *hardware*. Devido à modularidade do sistema, é possível modificar a interface de entrada do módulo caso o fornecimento de dados não seja feito por um processador, sem prejuízo ao funcionamento dos outros módulos.

Como os módulos que consomem os dados do *bitstream* fazem leituras de tamanho variável e não alinhadas a byte a arquitetura proposta para esse módulo armazena os dados

recebidos e encaminha os mesmos de forma serial.

O controle do fluxo de dados de entrada é feito através dos sinais *data_vld_i*, que indica se existe dado válido para a leitura, e *buffer_full_o*, que sinaliza se todos registradores estão em uso. Os módulos que necessitam do *bitstream* fazem leituras de tamanhos diferentes, por isso a saída de dados do módulo foi projetada para 1 bit por ciclo, com sinais de controle *bit_vld_o*, para sinalizar a existência de bit válido na saída, e *bit_req_i*, indicando o pedido de novo bit gerado por algum dos outros módulos. Foi incluído o sinal *byte_align_i*, que causa o alinhamento dos dados com o próximo *byte*, descartando bits se necessário. Esse alinhamento é controlado por um contador interno.

O módulo *Bitstream Buffer* controla a entrada do *bitstream* e fornecimento de *bits* para os outros módulos. A conexão *data_i* é usada para a entrada de dados do *bitstream*, com largura de 32 bits. Os sinais *data_vld_i* e *buffer_full_o* servem para o controle do fluxo de dados de entrada. O módulo possui 2 registradores de deslocamento de 32 bits para armazenar os dados do *bitstream* e gerar a saída de 1 bit *bit_o* a cada ciclo que *bit_req_i* está em nível alto.

5.1.2. Módulo Bitstream Decoder

O módulo *Bitstream Decoder* se encontra conectado diretamente ao *Bitstream Buffer* para realizar a leitura de dados do *bitstream*. Ele também possui uma interface de comunicação com o módulo de intermediário, que conecta os blocos de decodificação do espectro e reconstrução do áudio. Os coeficientes espectrais decodificados são enviados para o módulo intermediário onde são armazenados até o uso pelo Banco de Filtros.

Estão implementados no módulo uma máquina de estados que controla o processo de decodificação do *bitstream*, uma segunda máquina que realiza a comunicação com o módulo *Bitstream Buffer*

A máquina de estados de controle proposta na arquitetura do módulo inicia o processo de decodificação do *bitstream* lendo os primeiros 3 bits do mesmo para identificar o elemento sintático recebido. A escolha do próximo estado da máquina depende do valor contido nesses 3 bits. Caso o valor corresponda ao elemento SCE é gerado um sinal de controle para retirar o módulo decodificador de SCE do *reset* e a máquina avança para o estado *decode_sce*. Ela permanece nesse estado até que o módulo SCE termine a decodificação, quando então ela volta ao estado inicial para decodificar o próximo elemento.

O procedimento ao receber um elemento CPE é semelhante ao descrito para o SCE, mas o módulo que ativado é o decodificador de CPE. Assim como na decodificação do SCE, quando CPE termina a decodificação a máquina retorna ao estado inicial e procura o próximo elemento no *bitstream*.

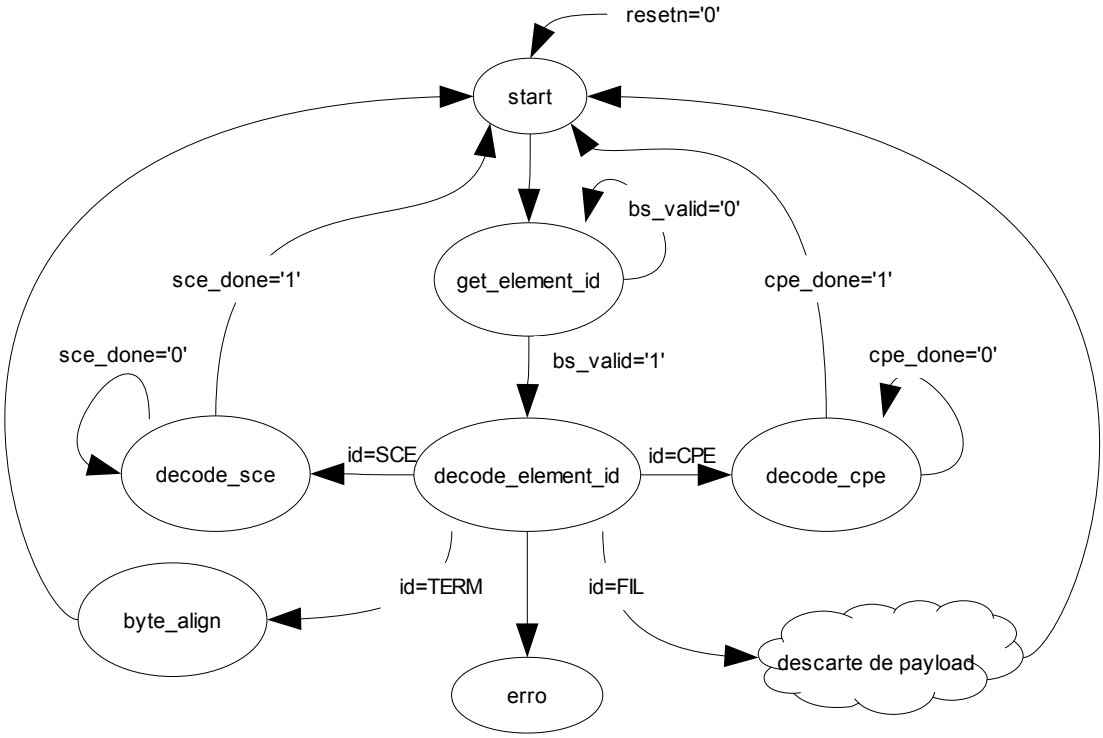


Figura 29: Diagrama de estados da máquina de controle de Bitstream Decoder sem detalhar descarte de payload

Sendo identificado um elemento sintático do tipo FIL, no próximo estado da máquina são lidos novos bits do *bitstream* para decodificar o número de bytes transportado pelo elemento. A decodificação desse valor começa com a leitura de 4 bits do *bitstream*. Se todos bits tiverem valor '1' são lidos mais 8 bits do *bitstream* e o valor 14 é somado a essa última leitura para determinar o tamanho do pacote, caso contrário o valor é dado pelos 4 bits lidos originalmente. Decodificado o tamanho do *payload* contido no elemento FIL, a máquina procede com o descarte desse elemento realizando a leitura do mesmo do *bitstream*. O descarte é controlado pelo uso de um contador inicializado com o tamanho do FIL que é decrementado a cada byte descartado até que chegue a 0.

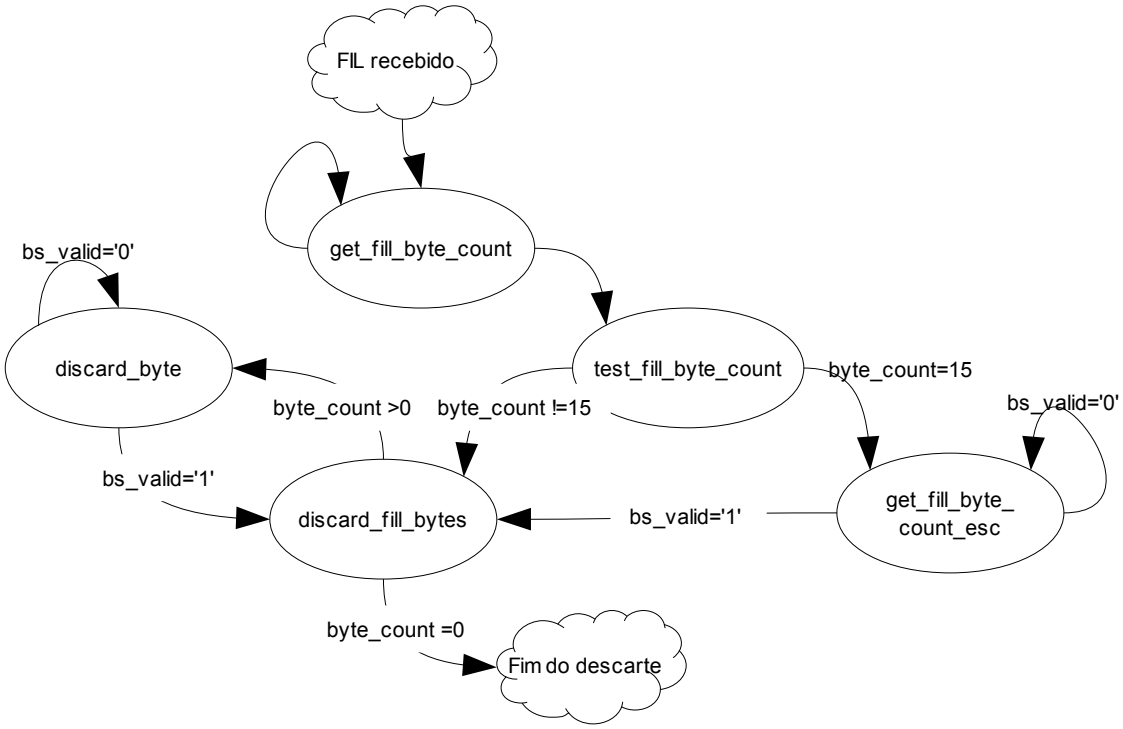


Figura 30: Detalhamento do descarte de bytes do elemento FIL

O elemento TERM força o alinhamento a byte do *bitstream* avançando para o estado *byte_align* e significa que o *frame* atual terminou. No ciclo seguinte do relógio a máquina retorna ao estado inicial pronta para decodificar o próximo *frame*.

Quando não é possível a decodificação de um elemento recebido devido ao mesmo não ser suportado pelo decodificador ocorre a transição para um estado de erro e um sinal para identificação do erro é gerado na saída do módulo.

Uma segunda máquina de estados controla a comunicação com *Bitstream Buffer* lendo quantidades variáveis de bits e armazenando em um registrador. Isso foi incluído nesse módulo para evitar que outros módulos tivessem que gerar controles semelhantes para leitura de palavras do *bitstream*. Os sinais de controle passados para essa máquina são *bs_read*, que inicia o processo de leitura do *bitstream*, e *bs_count*, que informa a quantidade de bits que deve ser lida. Esses sinais recebem a saída de multiplexadores com sinais provenientes dos outros módulos para que todos possam acessar o *bitstream* quando necessário.

A multiplexação do módulo ICS INFO também é realizada nesse módulo. Dois módulos podem acessar o ICS INFO: o módulo *CPE Decoder*, quando ambos canais forem codificados com a mesma configuração de janela, e o módulo *ICS Decoder*, quando as configurações de janela forem independentes no elemento CPE ou quando estiver decodificando um elemento SCE.

5.1.3. MÓDULO SCE DECODER

O módulo decodificador do elemento SCE sempre recebe o identificador *element instance tag* do *bitstream* e executa o decodificador de ICS uma única vez. Assim que o sinal de fim de execução do decodificador ICS é recebido o decodificador SCE comunica ao controle de *Bitstream Decoder* para que o processamento do próximo elemento possa ser iniciado. O SCE possui dado para reconstrução de um canal de áudio.

A arquitetura proposta possui uma máquina de estados que gera sinais para a leitura de dados do *bitstream* e para a ativação do módulo de decodificação do ICS. Os sinais de entrada usados para controle na máquina são o de ativação do módulo, fim da leitura do *bitstream* e fim da execução do decodificador de ICS.

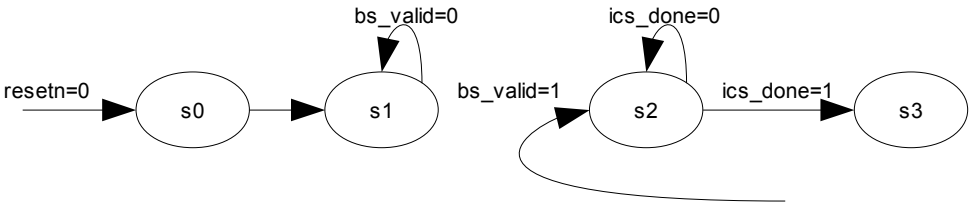


Figura 31: Diagrama de estados do decodificador SCE

Tabela 2: Sinais da máquina de estados do decodificador SCE

Sinal	Descrição
bs_read	Habilita leitura do *bitstream*
bs_count	Número de bits que devem ser lidos do *bitstream*
ics_resetn	Habilita decodificador ICS quando em nível 1
done	Informa o fim da execução do decodificador SCE
resetn	Habilita o decodificador SCE quando em nível 1
bs_valid	Informa que dados requisitados do *bitstream* são válidos
bs_data	Dados lidos do *bitstream*
ics_done	Informa fim da execução de decodificador ICS

No estado s1 a leitura do *bitstream* é iniciada e a máquina só avança para o estado s2 quando os 4 bits requisitados são válidos. No estado s2 o módulo decodificador de ICS é ativado e a máquina só avança para s3 quando é sinalizado o fim da decodificação do ICS. No estado s3 o sinal *done* é colocado em nível lógico alto. Sempre que o sinal *resetn* está em nível lógico 0 a máquina volta para o estado inicial s0.

5.1.4. MÓDULO CPE DECODER

O módulo decodificador do elemento CPE é ativado pelo *Bitstream Decoder*. Ele implementa uma máquina de estados que controla a ativação do módulo que decodifica o ICS. Os dados *element instance tag* e *common window* são extraídos do *bitstream* nesse módulo, e o parâmetro *common window* é usado para definir se o decodificador de ICS INFO deve ser ativado antes do decodificador ICS. Como o pacote com este elemento sintático possui dois canais de áudio, o decodificador ICS é executado duas vezes antes do fim da operação do bloco ser sinalizado.

A máquina de estados proposta para a decodificação do elemento CPE é mais complexa que a apresentada para o SCE pois os dados contidos no *bitstream* podem alterar o fluxo dos estados da máquina. Os sinais descritos para o módulo SCE na Tabela 2 também são encontrados no CPE além dos descritos na Tabela 3.

No estado s1 a máquina espera a leitura de 5 bits do *bitstream*, sendo 4 bits da identificação *element instance tag* e 1 do parâmetro *common window*. Conforme exposto na seção 4.2, o pacote CPE pode usar a mesma configuração de janela para os dois canais de áudio, o que resulta no envio de um único ICS INFO e é sinalizado pelo parâmetro *common window*.

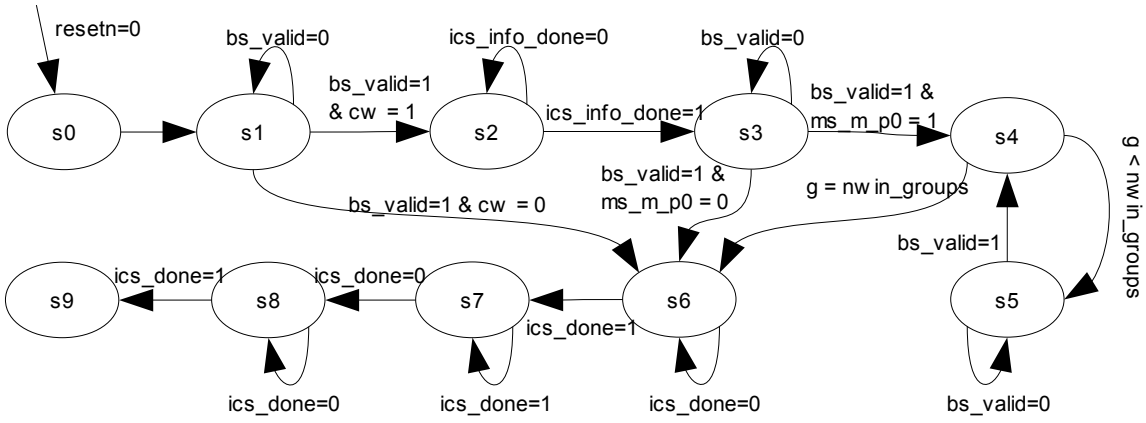


Figura 32: Máquina de estados do decodificador CPE

Quando *common window* tem valor '1', a máquina avança para o estado s2, no qual é ativada a decodificação do ICS INFO. Ao término da decodificação de ICS INFO, a máquina segue para o estado s3, onde é lido do *bitstream* o parâmetro *ms mask present*. Caso o bit

menos significativo de *ms mask present* seja '1', a máquina começa a decodificação dos parâmetros *ms used*, realizada no laço entre os estados s4 e s5. Terminada a recepção de *ms used*, o estado seguinte é s6.

Tabela 3: Sinais da máquina de estados do decodificador CPE

Sinal	Descrição
cw	Parâmetro *common window*, recebido do *bitstream* no estado s1
ics_info_done	Informa fim da execução do decodificador ICS INFO
ms_m_p0	Bit menos significativo do parâmetro *ms mask present*, recebido do *bitstream* no estado s3
nwin_groups	Parâmetro *num window groups* decodificado em ICS INFO
g	Saída de contador de 4 bits
max_sfb	Parâmetro *max sfb* decodificado em ICS INFO
ics_info_resetn	Habilita decodificador ICS INFO quando em nível 1

No caso de *common window* ser '0' no estado s1 ou *ms mask present* ser '0' no estado s3, a máquina avança diretamente ao estado s6. Entre os estados s6 e s9 a máquina realiza a decodificação de dois blocos ICS, um para cada canal.

5.1.5. Bloco ICS Decoder

O módulo decodificador de ICS realiza a recuperação dos dados de tabelas Huffman usadas na codificação, fatores de escala e coeficientes espectrais. Ao final da execução desse módulo o espectro do canal de áudio se encontra reconstruído e pronto para ser usado na IMDCT para gerar as amostras de áudio do *frame*. Esse módulo é usado na decodificação dos elementos SCE e CPE, e deverá ser usado em uma futura implementação do LFE.

A arquitetura proposta divide esse módulo em um bloco de controle da decodificação e blocos para decodificação de *Section Data*, *Scalefactor Data* e *Spectral Data*. A decodificação do ICS INFO não está incluída nesse módulo pois é compartilhada com o módulo CPE Decoder.

Algumas áreas de memória foram usadas nesse módulo para armazenar os dados extraídos de *Section Data* e *Scalefactor Data*. O valor dos fatores de escala varia entre 0 e 255, então 8 bits são suficientes para cada fator de escala. A quantidade de fatores de escala transmitidos depende da configuração da janela de transformada, da frequência de

amostragem do sinal e, no caso de janelas curtas, do agrupamento das janelas. Para os dados de *Section Data* são necessários outros dois blocos de memória para armazenar palavras de 4 bits representando a tabela de códigos Huffman. A quantidade de palavras que deve ser armazenada em um dos blocos é igual ao número de bandas de fatores de escala (uma tabela por banda) e no outro bloco depende do número de seções na janela.

A máquina de estados do bloco de controle gera sinais de controle para os módulos decodificadores de ICS INFO, *Section Data*, *Scalefactor Data* e *Spectral Data*. No estado s1 a máquina espera a leitura do parâmetro *global gain* do *bitstream*. O valor de 8 bits é registrado e a máquina avança. O próximo estado pode ser s2 ou s3, dependendo do valor do parâmetro *common window*, representado por *cw*. Se *cw* tem valor '0' significa que o bloco ICS INFO deve ser decodificado, então a máquina passa para o estado s2 e espera que ICS INFO seja extraído do *bitstream* e por fim passa ao estado s3. Caso *cw* apresente valor '1' a máquina muda direto para o estado s3, onde é iniciada a decodificação de *Section Data*. O passo seguinte, no estado s4, é a decodificação de *Scalefactor Data*. Quando *sf_data_done* passa a ter valor '1' a máquina avança para o estado s5, onde ela espera a leitura de 3 bits do *bitstream*. Os bits lidos sinalizam o uso das ferramentas *Pulse Coding*, *TNS* e *Gain Control*, que não são suportadas. Todos bits devem ser zero para que a decodificação continue normalmente, passando para o estado s6. Se a máquina passar para o estado s8 o erro pode ser verificado através dos sinais *tns_present*, *pulse_present* e *gain_ctrl_present*. No estado s6 a máquina espera o término da decodificação de *Spectral Data* e finaliza a máquina no estado s7.

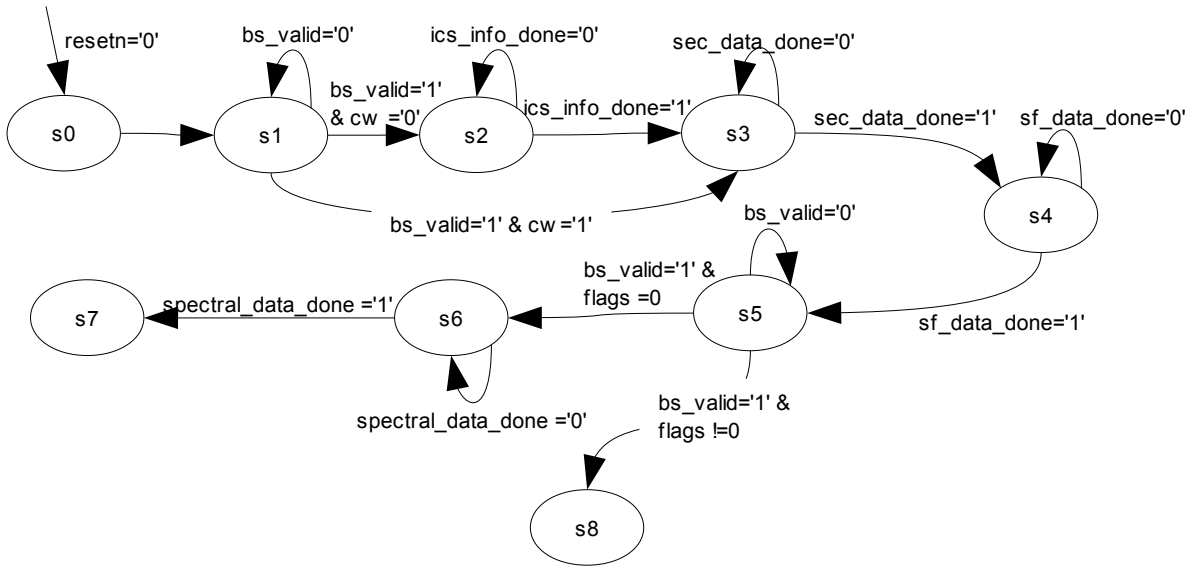


Figura 33: Diagrama de estados da máquina de controle de ICS Decoder

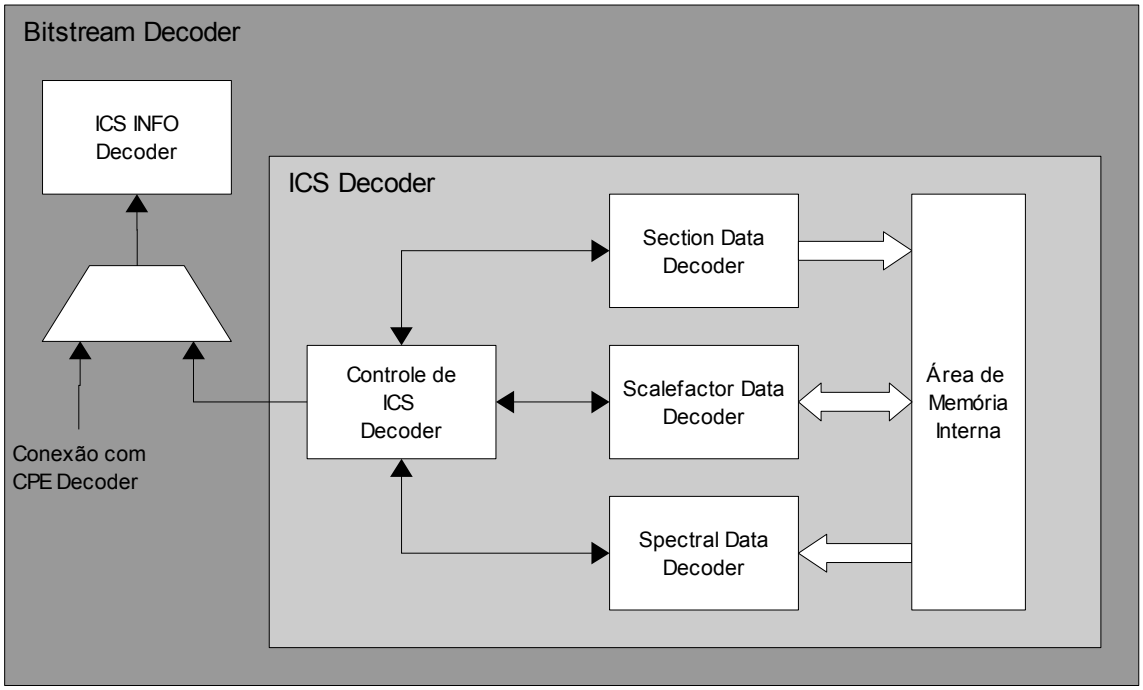


Figura 34: Diagrama com blocos internos de ICS Decoder e comunicação com ICS INFO

5.1.6. Módulo Decodificador ICS INFO

No módulo ICS INFO é realizada a extração de dados de configuração da janela de transformada associadas a um ou dois blocos ICS e consequentemente a um ou dois canais de áudio. Além disso extrai as informações de agrupamento das janelas no caso de uma sequência de janelas curtas.

A arquitetura proposta possui duas máquinas de estados sendo uma para a extração dos dados pertencentes ao bloco ICS INFO e a outra para o processamento dos dados recebidos e na determinação do agrupamento das janelas.

A extração dos dados de ICS INFO inicia e a máquina espera no estado s1 a leitura de 4 bits do *bitstream*. O bit menos significativo contém o valor de *window shape* e os 2 bits seguintes representam *window sequence*. O bit mais significativo não tem uso no padrão atual, é reservado para uso futuro. O processo varia de acordo com *window sequence* por isso no estado s2 o valor lido é usado para decidir o próximo estado.

O recebimento de *window sequence* com valor "10", que representa *Eight Short Sequence*, a máquina passa para o estado s3, onde é aguardada a leitura de 11 bits. Os 4 bits

mais significativos constituem o valor de *max sfb*, que informa o número de bandas de fatores de escala por janela. Os 7 bits restantes contém o parâmetro *scale factor grouping* que possui informações sobre o agrupamento das janelas. Os dois valores são armazenados em registradores e a máquina passa para o estado s4. Nesse estado é iniciada a segunda máquina de estados para a determinação do agrupamento das janelas. Assim que a segunda máquina termina sua execução a primeira passa para o estado s5 onde é indicado o término da mesma.

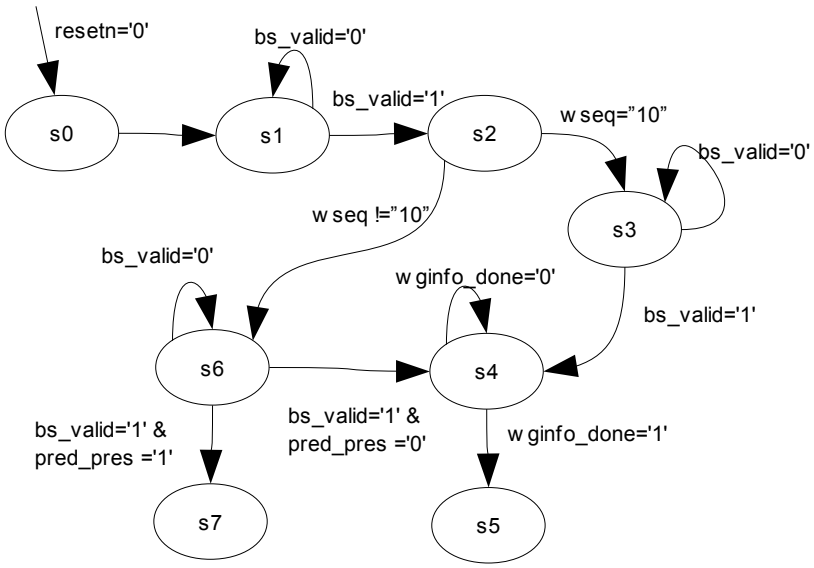


Figura 35: Diagrama de estados da máquina principal de ICS INFO

Quando *window sequence* tem qualquer valor diferente de "10", ou seja, a janela de transformada usada é longa, a máquina passa para o estado s6. Nesse estado é realizada a leitura de 7 bits do *bitstream*, sendo os 6 mais significativos o valor de *max sfb* e o bit restante um indicador do uso da ferramenta *Long Term Prediction,* representado pelo sinal *pred_pres* no diagrama. Como essa ferramenta não é suportada pelo AAC LC, se o bit indicar seu uso a máquina passa para o estado de sinalização de erro, s7. Caso *pred_pres* seja '0' a decodificação continua e a máquina passa para o estado s4. Aqui a máquina para o cálculo do dados de agrupamento é iniciada. Ao final da sua operação a primeira máquina também encerra, passando para o estado s5.

Na segunda máquina de estados do módulo, no estado inicial s0 o valor de *window sequence* é usado para decidir qual o próximo estado e valores de inicialização para alguns registradores. Para sequencias de janelas curtas o estado seguinte é o s1, onde a máquina continua no mesmo estado por 7 ciclos determinando o agrupamento das janelas. O número

de grupos de janelas, *num_win_groups*, é calculado em s1. No estado s2 e s3 os valores de *offset d*e cada seção em um grupo são calculados e gravados em uma memória, finalizando a máquina após esse processo. A cada transição de s3 para s2 o contador *g* é incrementado. No caso de janelas longas o segundo estado que a máquina assume é s4, onde valores de *offset* dos coeficientes espectrais das seções são armazenados em uma memória. Após esse procedimento a máquina é finalizada.

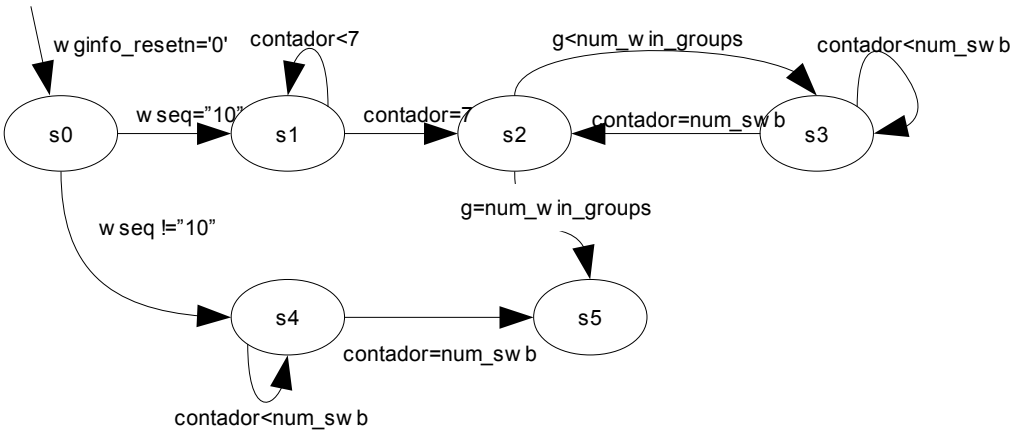


Figura 36: Diagrama de estados da máquina secundária em ICS INFO

5.1.7. Módulo Scalefactor Decoder

O módulo decodificador dos fatores de escala realiza a extração dos fatores de escala contidos no *bitstream* usando a tabela Huffman especificada no apêndice 4.A de (ISO/IEC, 2005). A busca do valor correto foi feita usando um algoritmo de busca binária baseado no código C do FAAD, um decodificador AAC o*pen source*.

A arquitetura proposta implementa o módulo em duas partes: uma máquina de estados que controla o processo e uma segunda máquina que realiza a decodificação Huffman da diferença transmitida. Duas tabelas de 241 valores, uma com palavras de 7 bits e a outra com 6 bits, estão presentes nesse módulo para a decodificação Huffman dos fatores de escala.

A máquina de controle carrega os sinais de entrada no estado s1. Dois laços aninhados são executados envolvendo os estados seguintes para que sejam decodificados os fatores de escala de todas bandas de fatores de escala em todos grupos de janelas. No estado s2 se encontra o controle do laço externo, onde a máquina é finalizada quando o contador *g* atinge o valor de *num_win_groups*. O controle do laço interno, que percorre as bandas de fatores de

escala, é realizado no estado s3. Quando o contador *sfb* atinge o valor *max_sfb* a máquina passa para o estado s10 onde o contador *sfb* é zerado e *g* é incrementado para então voltar ao estado s2.

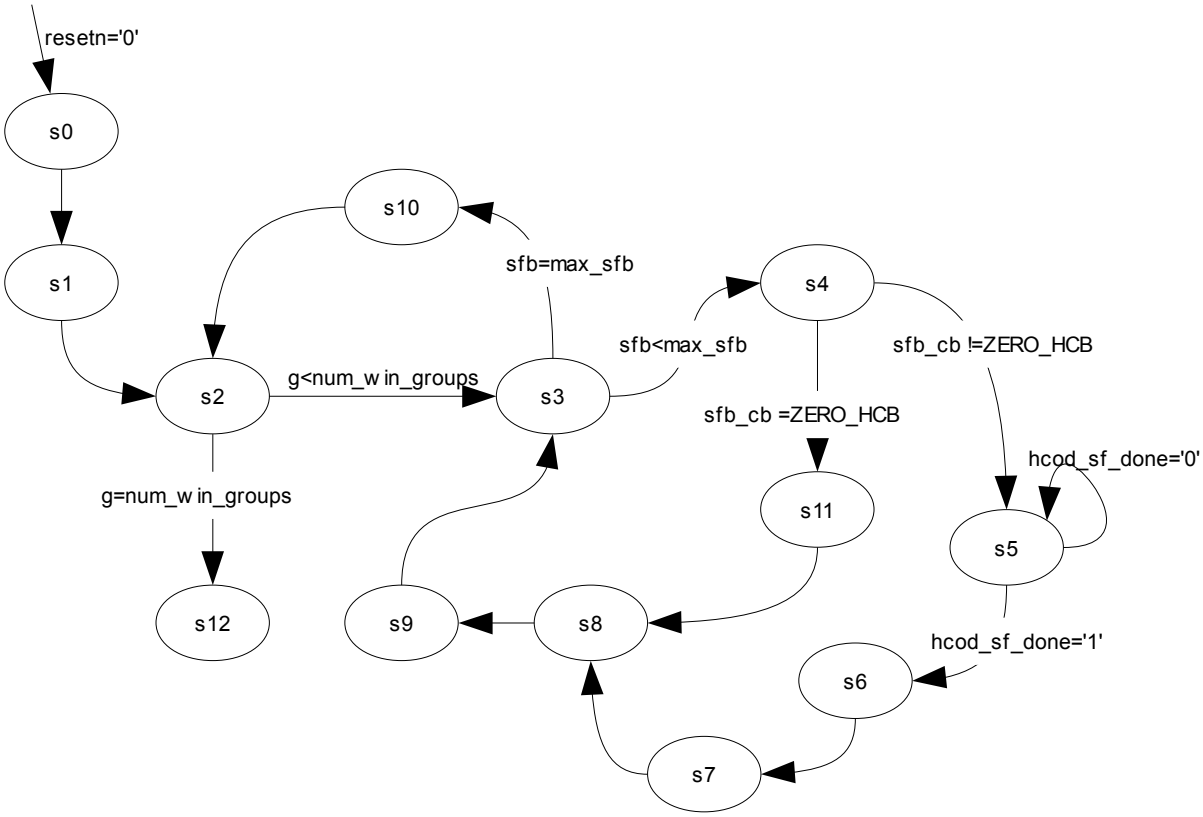


Figura 37: Diagrama de estados da máquina de controle de Scalefactor Decoder

No estado s4 é lido da memória o valor da tabela Huffman usada pra os coeficientes espectrais. Caso seja a tabela ZERO_HCB a máquina passa para o estado s11 com a finalidade de escrever o valor 0 na memória de fatores de escala. Se a tabela for qualquer outra a máquina passa para o estado s5, onde a decodificação Huffman é realizada. Os estados restantes são usados para calcular o valor do fator de escala a partir da diferença decodificada do *bitstream* e escrever o resultado em memória. No estado s9 o contador *sfb* é incrementado.

Esse módulo possui interface com um bloco de memória, onde são escritos os valores recuperados dos fatores de escala para que possam ser usados no módulo de *Spectral Data*. O endereçamento da memória é gerado a partir dos contadores *g* e *sfb* usados no bloco de controle da decodificação.

A decodificação Huffman é realizada por uma máquina de estados que lê um bit do

bitstream, atualiza o valor de *offset* usando a tabela correspondente ao bit lido (tabela 0 ou tabela 1) e por fim verifica testando o valor da tabela 1 na posição *offset* (estado s1) se os bits recebidos já formam um código válido. Uma operação de soma é usada para calcular o novo valor de *offset*. O valor decodificado é encontrado na posição determinada por *offset* da tabela 0. Os estados s2 e s3 realizam a leitura de um bit do *bitstream*.

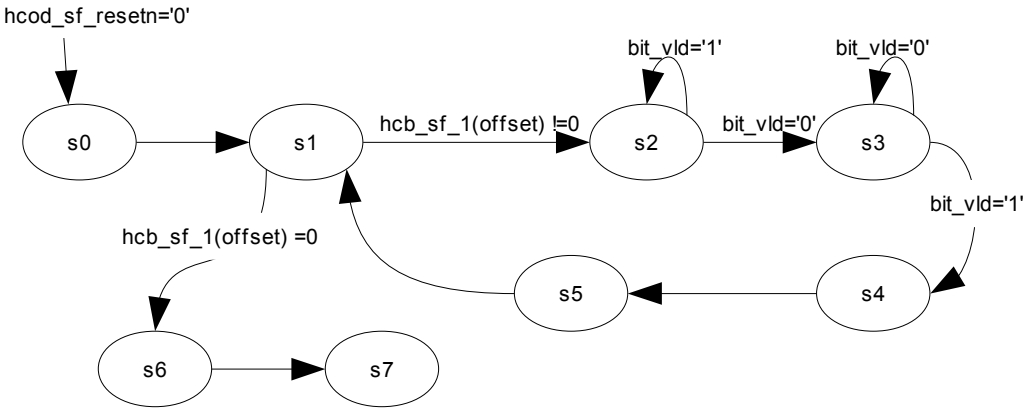


Figura 38: Diagrama de estados da decodificação Huffman

5.1.8. Módulo Spectral Data Decoder

Os coeficientes espectrais quantizados são decodificados diretamente do *bitstream* usando tabelas Huffman. Como já foi citado anteriormente, a tabela usada em cada seção pode ser diferente para otimizar a codificação. O módulo usa os dados recebidos em *Section Data* para determinar a tabela usada em cada seção. O *Spectral Data Decoder* é composto de um bloco que realiza a decodificação Huffman dos coeficientes espectrais, outro que executa a quantização inversa e um terceiro que aplica os ganhos definidos pelos fatores de escala. O módulo de decodificação Huffman está conectado a um bloco de memória, usado para armazenar os coeficientes espectrais ainda quantizados. Essa área pode armazenar até 512 palavras de 28 bits.

Dentro do bloco de decodificação Huffman, cada tabela está implementada em um módulo separado que tem uma pequena máquina de estados para fazer a leitura dos bits do e verificar se os dados recebidos formam uma palavra válida.

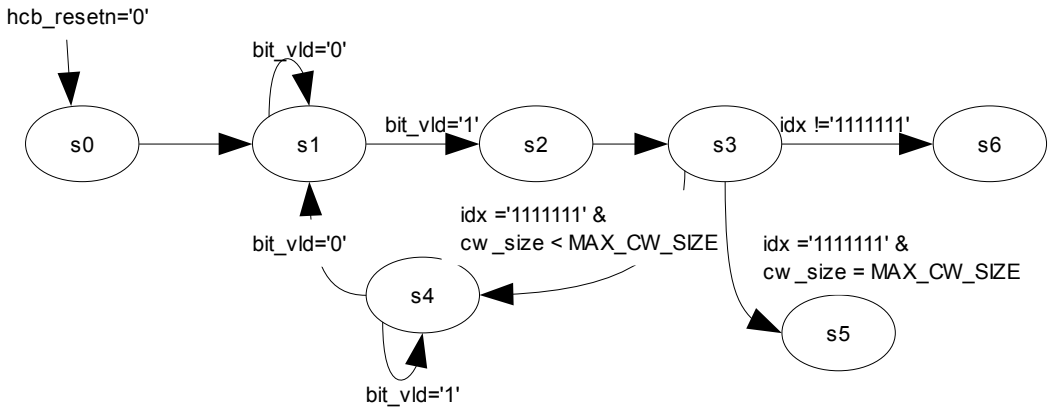


Figura 39: Diagrama de estados da decodificação Huffman do espectro

No estado s1 é realizada a leitura de um bit do *bitstream*. No estado s2 o novo bit lido é concatenado aos anteriores e *cw_size*, que representa o número de bits já lidos é incrementado. Em s3 é testado o valor de *idx*, que é o sinal onde o valor decodificado é disponibilizado. Caso seja diferente de 127 (para o caso da tabela 1 que foi usada como base para gerar a Figura 30, para outras o valor a ser comparado é outro) significa que foi encontrado um código válido e a decodificação termina. Se *idx* for 127 é verificado se *cw_size* já atingiu o valor limite para a tabela em questão e é tomada a decisão de buscar o próximo bit ou ir para o estado de falha s5.

A partir do *idx* são calculados os 2 ou 4, depende da tabela Huffman em uso, coeficientes espectrais quantizados.

A máquina de estados que controla a operação do módulo completo inicia o processo realizando a extração de todos coeficientes quantizados para o *frame* do canal sendo decodificado. Isso é feito usando três laços. O mais externo garante que todos grupos de janelas presentes no bloco de dados do canal serão processados usando *num_window_groups* para controle. O segundo laço abrange todas seções da janela em processamento. O laço mais interno passa por todas posições de coeficientes espectrais da seção. Todos os valores são escritos na área de memória reservada para o módulo, para então se dar início ao processo de quantização inversa e aplicação dos fatores de escala.

A aplicação dos fatores de escala está conectada diretamente ao bloco de quantização inversa para que as operações sejam realizadas em sequência, sem necessidade de utilizar área de memória para armazenar valores intermediários. Após a passagem pela quantização inversa o coeficiente espectral passa pela aplicação de ganho dos fatores de escala e é enviado

para o bloco de memória intermediária.

5.1.8.1 Quantização Inversa

A arquitetura proposta para o bloco de quantização inversa implementa uma aproximação da equação vista na seção 4.4. Esse método foi baseado nas arquiteturas apresentadas em (HEE, 2007) e (TSAI, 2009). Como o valor absoluto máximo de um coeficiente quantizado é 8191, deve ser possível obter o resultado da função de quantização inversa para a faixa de valores entre 0 e 8191. O algoritmo propõe que os valores de entrada sejam divididos em 3 faixas: a primeira de 0 a 255, a segunda de 256 a 2047 e a terceira de 2048 a 8191. Dependendo da região em que se encontra o número de entrada, uma função diferente é usada.

Para a primeira faixa de valores é usado o valor direto de $x^{4/3}$ que no módulo foi implementado em uma tabela com 256 palavras de 16 bits. Para a segunda faixa o valor é aproximado através da equação (5.1), onde $f(x)$ é um valor obtido na tabela e a divisão de x é arredondada para baixo.

$$x^{\frac{4}{3}} \approx 2 \cdot \left(f\left(\frac{x}{8}+1\right) - f\left(\frac{x}{8}\right) \right) \cdot \mathrm{rem}\left(\frac{x}{8}\right) + f\left(\frac{x}{8}\right) \cdot 16, \quad \text{para } 256 \le x < 2048 \tag{5.1}$$

O resultado para valores na terceira faixa é obtido por (5.2).

$$x^{\frac{4}{3}} \approx 2 \cdot \left(f\left(\frac{x}{64}+1\right) - f\left(\frac{x}{64}\right) \right) \cdot \mathrm{rem}\left(\frac{x}{64}\right) + f\left(\frac{x}{64}\right) \cdot 256, \quad \text{para } 2048 \le x < 8192 \tag{5.2}$$

O sinal de entrada do bloco quantizador inverso é usado para gerar o endereçamento da tabela, sendo usado um multiplexador para selecionar entre o endereço das três faixas de valores. A tabela tem como saída o valor correspondente a posição p e o correspondente a posição $p+1$.

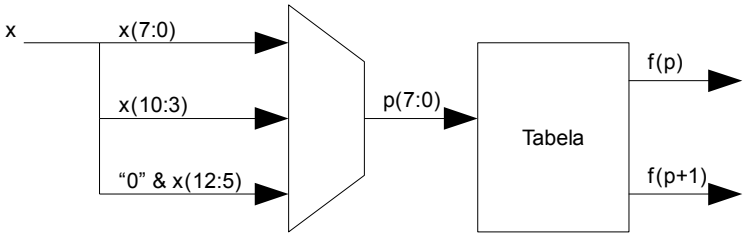


Figura 40: Diagrama do acesso à tabela de quantização inversa

No caso do valor de x se encontrar na primeira faixa, entre 0 e 255, a saída da tabela $f(p)$ é o resultado da operação. Para as faixas restantes as operações descritas em (5.1) e (5.2) são implementadas usando um multiplicador, dois somadores e operações de deslocamento. Os valores de saída desse módulo são representados por um bit de sinal, 18 bits de parte inteira e 5 bits para a parte fracionária.

5.1.8.2 Aplicação de Fatores de Escala

O módulo de aplicação dos ganhos calculados através dos fatores de escala implementa a operação descrita na seção 4.5. A arquitetura em *hardware* possui um *barrel shifter* capaz de deslocamentos de até 31 bits para a esquerda ou direita em um ciclo. Além disso são usados um multiplicador e um somador para o cálculo do coeficiente espectral.

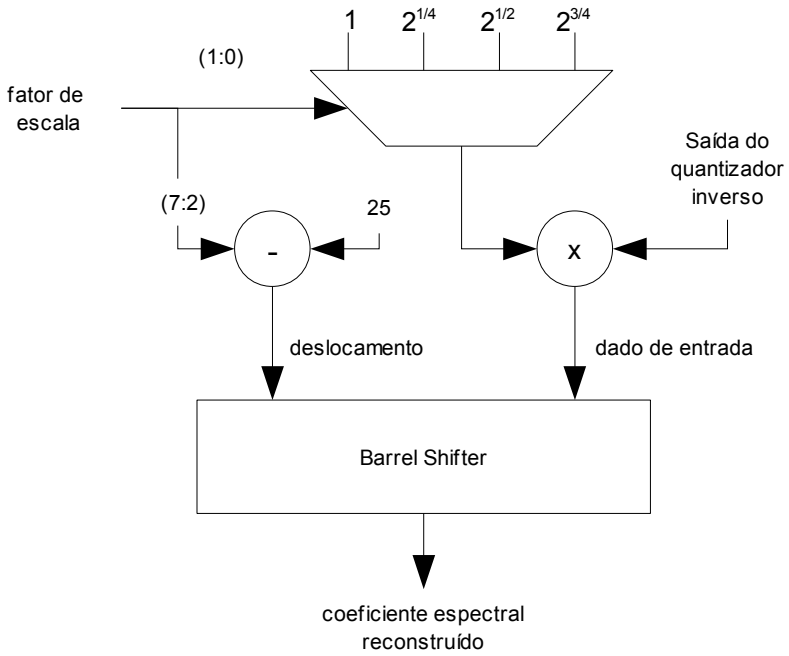


Figura 41: Diagrama da aplicação de fatores de escala

O dado de entrada desse módulo possui 24 bits e o de saída 32 bits. Desses 32 bits um representa o sinal e 6 a parte fracionária.

5.2. Comunicação entre blocos

O módulo responsável pela comunicação entre o bloco de decodificação espectral e o

de reconstrução de áudio possui uma grande área de memória para armazenar todos coeficientes espectrais de um *frame*. São 1024 palavras de 32 bits, totalizando uma área de 32768 bits. Além disso, possui registradores para armazenar os dados auxiliares de formato e tipo da janela, necessários para configurar os blocos de transformada inversa e janelamento do módulo posterior.

A máquina de estados que controla a operação do módulo tem somente dois estados. Ela inicia pronta para receber os coeficientes e escrevê-los na memória, com os sinais de controle indicando que a memória está livre. Assim que o módulo de decodificação dos coeficientes espectrais informa que terminou de processar o *frame*, a máquina passa para o estado em que as suas saídas de controle indicam que a memória contém dados válidos. A máquina volta para o estado inicial quando o bloco de reconstrução do áudio indica que terminou a leitura dos dados.

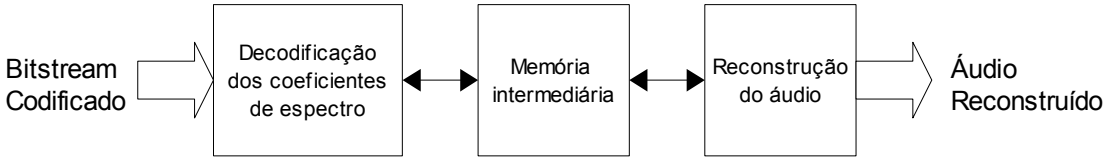


Figura 42: Área de memória entre blocos do decodificador

O uso de uma área de memória para separar os blocos de decodificação do espectro e reconstrução do áudio possibilita a execução em paralelo de ambos, implementando uma espécie de *pipeline*. Enquanto as amostras de áudio de um *frame* são reconstruídas o espectro do próximo *frame* é decodificado.

5.3. Reconstrução do áudio

A reconstrução das amostras de áudio a partir dos dados espectrais e auxiliares decodificados anteriormente é dividida em três módulos: transformada inversa, janelamento e sobreposição.

Figura 43: Diagrama de etapas da IMDCT

5.3.1. Transformada Inversa

Um *frame* compatível com o SBTVD pode conter uma única janela com 1024 coeficientes espectrais ou oito janelas consecutivas com 128 coeficientes cada, totalizando 1024. Sendo assim, a arquitetura proposta trata sempre 1024 palavras de entrada, diferenciando internamente o seu processamento de acordo com o tipo de janela usado na compressão.

O cálculo direto da IMDCT utiliza $N^2/2$ multiplicações e $N^2/2$ somas, o que é muito dispendioso para implementação no decodificador de áudio. Existem algoritmos rápidos propostos para realizar essa operação que diminuem consideravelmente a necessidade de multiplicações e somas, baseados em uma *Inverse Fast Fourier Transform* (IFFT) ou em uma *Discrete Cosine Transform* (DCT).

A arquitetura da IMDCT foi baseada no algoritmo apresentado por (DUHAMEL, 1991) e implementado em um FPGA por (DU, 2008), no qual a operação é dividida em três etapas. A mais complexa é uma *Inverse Fast Fourier Transform* (IFFT) de N/4 pontos, onde N é o número de pontos do vetor de saída da IMDCT. Em (DU, 2008) são utilizados 4 multiplicadores, enquanto na arquitetura proposta é usado somente um com a finalidade de economizar área, diminuindo a velocidade de processamento. Na implementação proposta N pode assumir o valor 2048 ou 256, tamanhos de janela determinados em (ISO/IEC, 2005) para o perfil AAC LC.

A IMDCT desenvolvida possui três módulos, onde cada um realiza uma etapa diferente do processo e seu funcionamento não é simultâneo. Essa característica favorece o compartilhamento de recursos críticos, como multiplicadores e memória. O bloco completo consome um multiplicador com entradas de 32 bits e uma área total de memória para 1024 palavras de 32 bits. A memória é organizada em dois blocos de 512 palavras, sendo um deles usado para armazenar a parte real e o outro para a parte imaginária dos dados complexos. A comunicação dos dados entre os módulos de Pré processamento, IFFT e Pós processamento ocorre através das áreas de memória do módulo IMDCT, cujo compartilhamento é gerenciado pelo bloco de controle da IMDCT.

O controle da IMDCT possui uma máquina de estados responsável por gerar os sinais para habilitar cada módulo interno, para liberar o controle do módulo de memória intermediária, e selecionar o módulo que tem acesso aos recursos compartilhados. O

chaveamento dos recursos compartilhados é feito através de multiplexadores que selecionam os sinais de entrada para o multiplicador e blocos de memória.

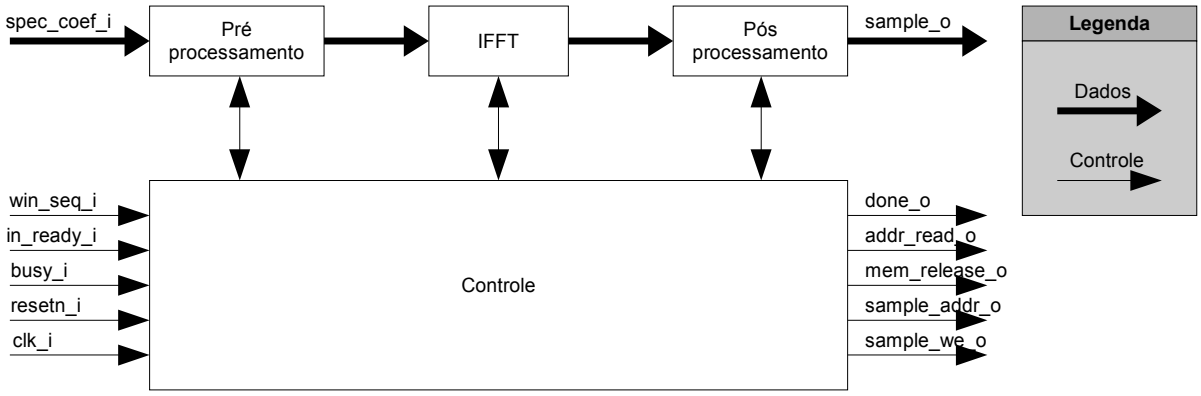


Figura 44: Diagrama de blocos da transformada inversa

5.3.1.1 Pré Processamento

Na primeira etapa do cálculo, realizada no módulo de Pré processamento, o vetor de coeficientes espectrais *X* armazenados no bloco de memória intermediário é lido e modificado de acordo com (5.3). O vetor resultante *Y* é guardado na memória interna da IMDCT para uso pela IFFT, separado em parte real e parte imaginária.

$$Y(k)=\left(-X(2k)+jX\left(\frac{N}{2}-1-2k\right)\right)e^{j\left(\frac{2\pi k}{N}+\frac{\pi}{4N}\right)},\quad 0\leqslant k<\frac{N}{4} \tag{5.3}$$

O cálculo de um valor de saída da equação (5.3) consome 4 ciclos de processamento do módulo de pré processamento. Isso se deve ao uso de um único multiplicador para realizar a multiplicação de dois valores complexos, operação que precisa ser quebrada em 4 multiplicações reais, uma por ciclo. Os valores de seno e cosseno necessários foram calculados e implementados no módulo em 4 tabelas com palavras de 32 bits, 2 com 512 palavras para uso com janela longa e 2 com 64 palavras para janela curta.

Este módulo possui uma máquina de estados interna que, após processar os 1024 coeficientes espectrais de entrada, gera um sinal indicando o fim do processamento. O controle do módulo IMDCT, ao receber esse sinal, desabilita o pré processamento, habilita o módulo IFFT e sinaliza que a memória intermediária pode ser sobrescrita.

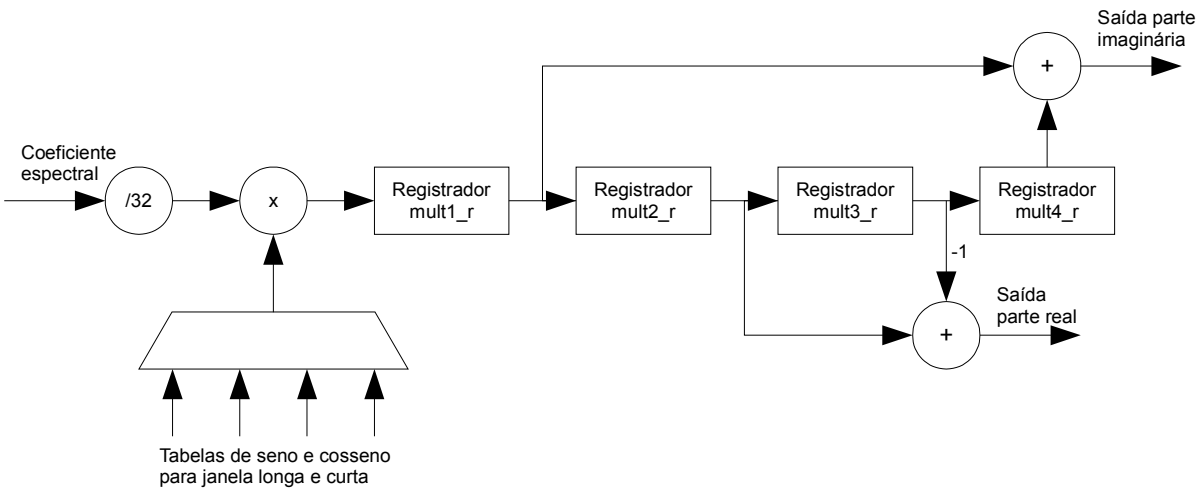


Figura 45: Fluxo de operação do pré processamento

A lógica que gera os endereços para a leitura dos dados da memória intermediária é selecionada de acordo com o tamanho da janela da transformada Quando a janela é longa o processo é realizado uma vez com *k* variando de 0 até 511. Para janelas curtas o processo é repetido 8 vezes com *k* variando de 0 até 63 e a cada repetição é acrescentado um deslocamento de 128 posições ao endereço de leitura gerado, varrendo as 1024 posições da memória intermediária.

5.3.1.2 IFFT

O módulo IFFT foi desenvolvido tomando como base o algoritmo de decimação em frequência. Nesse método o cálculo da Transformada Discreta de Fourier (DFT) de M pontos (5.4) é separado em dois somatórios, um para as amostras pares e outro para as ímpares (5.5) (OPPENHEIM, 1998).

$$X[k]=\sum_{n=0}^{M-1} x[n]W_M^{nk},\ \ k=0,1,...,M-1 \tag{5.4}$$

$$\text{onde } W_M^{nk}=e^{-j\left(\frac{2\pi}{M}\right)nk}$$

$$X[k]=\sum_{n\ \text{par}} x[n]W_M^{nk}+\sum_{n\ \text{ímpar}} x[n]W_M^{nk} \tag{5.5}$$

Através da manipulação da expressão (5.5) é possível transformar a operação em 2 DFTs de N/2 pontos (5.6). Essa simplificação pode ser feita recursivamente até que se atinja a

simplicidade desejada para o cálculo, onde o limite é uma DFT de 2 pontos (OPPENHEIM, 1998).

$$X[k]=\sum_{r=0}^{M/2-1} x[2r]W_{M/2}^{rk}+W_M^k\sum_{r=0}^{M/2-1} x[2r+1]W_{M/2}^{rk} \tag{5.6}$$

A arquitetura proposta para a IFFT possui um módulo para calcular a IFFT de 64 pontos, outro para a IFFT de 512 pontos e um para controle da operação. Os módulos são chaveados de acordo com o tamanho da janela da IMDCT.

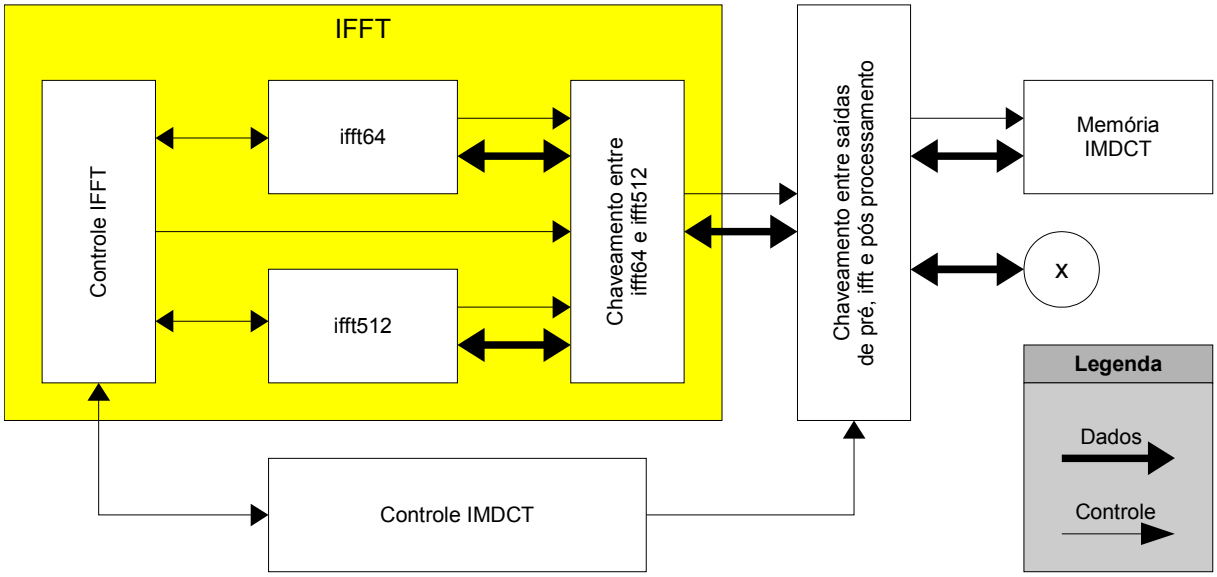


Figura 46: Diagrama do módulo IFFT

O cálculo completo da IFFT é dividido em $\log_2$M estágios, onde M é o número de pontos da IFFT. Os estágios são compostos por M/2 operações básicas, chamadas de *butterfly*. Cada estágio recebe M valores de entrada complexos resultantes do módulo de pré processamento no caso do primeiro estágio, ou do estágio anterior, e gera M valores de saída complexos que são armazenados para uso com o próximo estágio do cálculo. O *butterfly* é representado pelas equações (5.7) e (5.8).

$$x_s[p]=x_{s-1}[p]+x_{s-1}[q] \tag{5.7}$$

$$x_s[q]=\left(x_{s-1}[p]-x_{s-1}[q]\right)W_M^{-r} \tag{5.8}$$

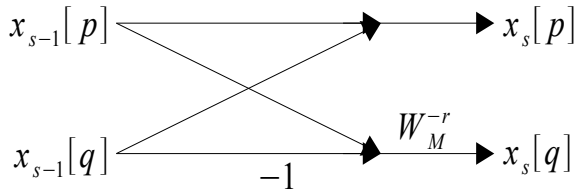


Figura 47: Diagrama de um butterfly

A ordem das operações e organização dos dados armazenados foi escolhida de acordo com o algoritmo *in place* com o objetivo de economizar área de memória, pois este possibilita o uso da mesma área em que se encontram os dados de entrada para guardar os cálculos intermediários e armazenar os resultados da IFFT. Cada estágio possui uma máquina de estados que controla a geração de endereço e o sinal de escrita/leitura para os blocos de memória da IMDCT, além de habilitar a escrita nos registradores internos.

O módulo de IFFT possui duas tabelas de valores de seno e duas tabelas de cosseno que são usadas para a operação do *butterfly*. As tabelas usadas para um *butterfly* da IFFT de 64 pontos são compostas de 32 palavras de 9 bits, enquanto para o cálculo da IFFT de 512 pontos as tabelas tem 256 palavras de 9 bits.

Após a execução de todos estágios do cálculo da IFFT, as saídas que se encontram nos blocos de memória da IMDCT são reordenadas para que sejam lidas pelo bloco de pós processamento da transformada.

A máquina de controle da IFFT está ilustrada na figura 48. O estado *start* é o estado em que a máquina inicia após o *reset*. No primeiro ciclo de operação o sinal de entrada *long_window_i* é usado para decidir o próximo estado. Quando o *frame* de áudio apresenta uma janela longa a máquina avança para o estado *ifft512*, no qual é gerado um sinal para ativar o módulo da IFFT de 512. O estado permanece o mesmo até o final da execução do módulo de 512 pontos, verificado através do sinal *ifft512_done*. para avançar ao estado *done*.

Em *frames* compostos por janelas curtas é necessário que o módulo IFFT de 64 pontos seja executado 8 vezes para processar todas as janelas. Ao iniciar a máquina de estados o sinal *long_window_i* deverá se encontrar no nível lógico zero, indicando *ifft64* como próximo estado. No estado *ifft64* o módulo IFFT de 64 pontos é ativado e o avanço da máquina é interrompido até que seja sinalizado o fim da execução do módulo. O estado seguinte habilita o incremento de um contador interno de 4 bits. No estado *page_test* é verificado se o módulo

IFFT curto já foi executado 8 vezes para então finalizar o processamento do *frame* ou voltar para o estado ifft64.

O endereçamento para leitura e escrita na memória depende do comprimento de janela usado. No caso de janela longa, o endereço completo de 9 bits é gerado dentro do módulo IFFT de 512 pontos. Para a janela curta, como cada execução da IFFT de 64 pontos usa somente 64 valores de cada área de memória, o endereço é gerado concatenando os 3 bits menos significativos do contador de controle da IFFT com 6 bits gerados dentro do módulo.

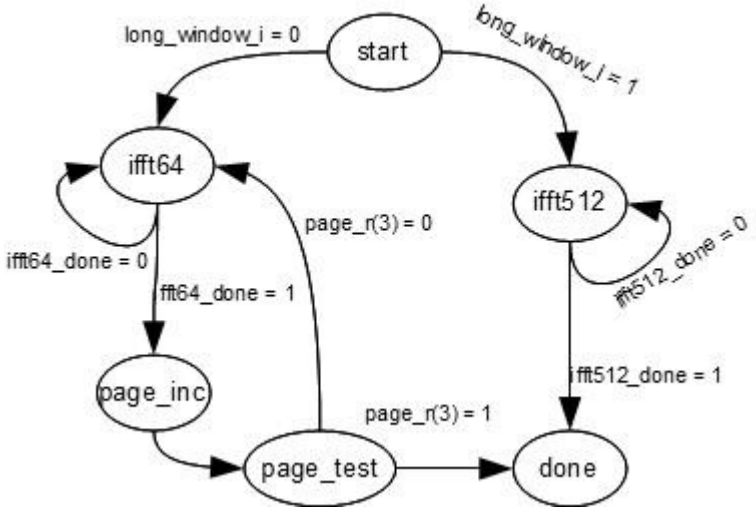


Figura 48: Diagrama de estados de controle IFFT

5.3.1.3 Pós Processamento

O módulo de Pós processamento utiliza as mesmas tabelas de senos e cossenos do módulo de Pré processamento para o primeiro passo, que é a execução da equação (5.9) onde *X* é o vetor de valores complexos resultante da IFFT que se encontra armazenado nos blocos de memória da IMDCT.

$$Y(k) = X(k)\, e^{j\left(\frac{2\pi k}{N} + \frac{\pi}{4N}\right)}, \quad 0 \le k \le \frac{N}{4} \tag{5.9}$$

Uma máquina de estados controla a seleção dos coeficientes corretos e o processo de leitura e escrita na memória. Os valores de *Y* são armazenados nos blocos de memória na

mesma posição dos valores de entrada. Essa operação requer uma multiplicação de dois valores complexos, que é dividida em 4 multiplicações reais e 2 somas.

O segundo passo do Pós processamento é a leitura dos dados ordenadamente para o uso no módulo de janelamento. Uma segunda máquina de estados controla esse procedimento, onde os valores complexos são separados e ordenados para gerar N saídas. O processo é dividido em 4 etapas diferentes que originam N/4 pontos, onde *x* é o valor armazenado em memória e *y* é a saída do módulo. Cada etapa possui duas equações, uma para os pontos pares e outra para os ímpares. A primeira parte é representada por (5.10).

$$\begin{aligned} y(2n) &= \mathrm{Im}\left[x\left(\frac{N}{8}+n\right)\right],\ \ 0 \le n < \frac{N}{8} \\ y(2n+1) &= \mathrm{Re}\left[x\left(\frac{N}{8}-n-1\right)\right],\ \ 0 \le n < \frac{N}{8} \end{aligned} \tag{5.10}$$

A segunda etapa é dada por (5.11).

$$\begin{aligned} y\left(\frac{N}{4}+2n\right) &= \mathrm{Re}[x(n)],\ \ 0 \le n < \frac{N}{8} \\ y\left(\frac{N}{4}+2n+1\right) &= -\mathrm{Im}\left[x\left(\frac{N}{4}-n-1\right)\right],\ \ 0 \le n < \frac{N}{8} \end{aligned} \tag{5.11}$$

A terceira parte é representada por (5.12).

$$\begin{aligned} y\left(\frac{N}{2}+2n\right) &= \mathrm{Re}\left[x\left(\frac{N}{8}+n\right)\right],\ \ 0 \le n < \frac{N}{8} \\ y\left(\frac{N}{2}+2n+1\right) &= -\mathrm{Im}\left[x\left(\frac{N}{8}-n-1\right)\right],\ \ 0 \le n < \frac{N}{8} \end{aligned} \tag{5.12}$$

A quarta etapa do processo é dada por (5.13).

$$\begin{aligned} y\left(\frac{3\mathrm{N}}{4}+2n\right) &= -\mathrm{Im}[x(n)],\ \ 0 \le n < \frac{N}{8} \\ y\left(\frac{3\mathrm{N}}{4}+2n+1\right) &= \mathrm{Re}\left[x\left(\frac{N}{4}-n-1\right)\right],\ \ 0 \le n < \frac{N}{8} \end{aligned} \tag{5.13}$$

5.3.2. Módulo de Janelamento

O módulo de janelamento possui uma entrada *clk_i* que fornece o relógio para os seus componentes e um sinal de *reset_i* que serve para reiniciar as máquinas de estados e registradores internos. Além disso, os parâmetros de configuração *window shape* e *window sequence* do bloco de áudio em decodificação são passados através dos sinais de entrada *window_shape_i* e *window_seq_i* respectivamente. A comunicação dos dados de entrada é feita através de 3 sinais: *din_ready_i* que indica a presença de dado válido na entrada, *din_i* que é o dado de entrada recebido do módulo da transformada e *din_n_i* que é a posição do dado de entrada no bloco de áudio completo. O sinal *din_i* tem 32 bits e *din_n_i* tem 11 bits.

O fluxo dos dados dentro do módulo depende do parâmetro *window sequence*. No caso do bloco de áudio estar codificado com uma das três configurações que usam a janela longa (*only long*, *long start* ou *long stop*) a única operação realizada sobre os dados de entrada é a multiplicação por um coeficiente de janela, representada no diagrama pelo bloco janelamento.

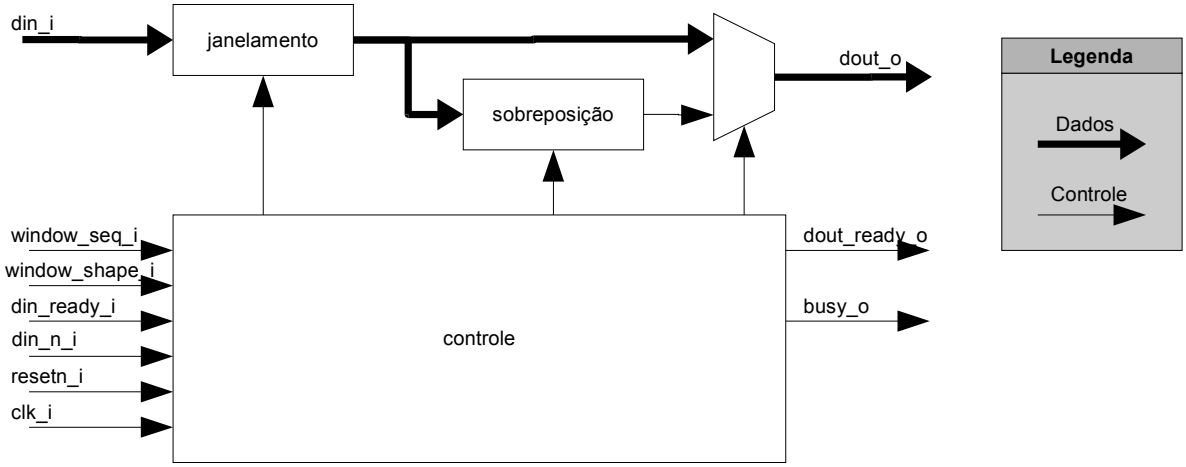


Figura 49: Diagrama geral do módulo de janelamento

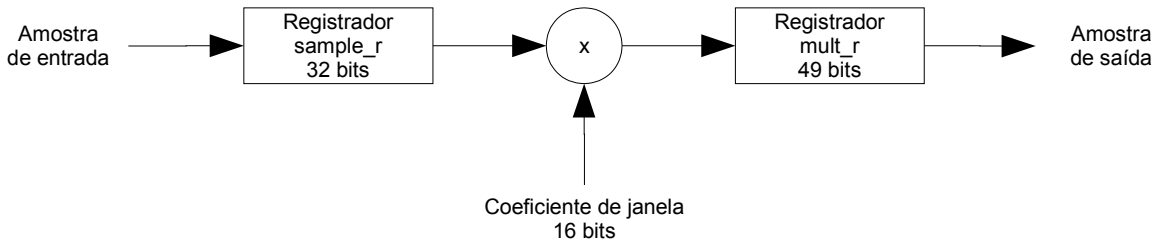


Figura 50: Fluxo de dados da multiplicação no bloco de janelamento

O processamento do conjunto de janelas curtas necessita que o fluxo de dados seja modificado para acomodar a sobreposição das janelas do bloco de áudio. Duas cadeias de

registradores e um somador são acrescidos ao final do fluxo de janela longa. Cada cadeia de registradores tem 128 posições, possibilitando a sobreposição de 50% de cada janela curta. No diagrama geral esse é o bloco sobreposição.

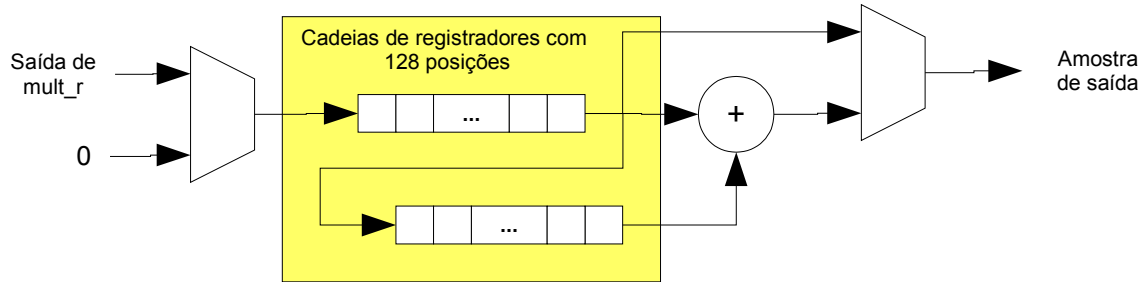


Figura 51: Sobreposição de janelas curtas

Os coeficientes de janelamento usados na multiplicação estão armazenados em 4 tabelas dentro do bloco de janelamento, em um módulo denominado *win_rom*. A escolha da tabela é feita pela combinação do parâmetros *window sequence* e *window shape*. O sinal *window_shape_i* determina qual o tipo da função de janela (senoidal ou KBD, conforme seção 4.6) enquanto o sinal *window_seq_i* é usado para definir o arranjo das janelas.

Como as funções de janela são simétricas, as tabelas contém somente a primeira metade dos coeficientes de cada janela. Sendo assim temos duas tabelas com 128 coeficientes e outras duas com 1024 coeficientes onde cada coeficiente tem 16 bits. A segunda metade das tabelas é gerada através da modificação do endereço de acesso realizada no módulo *win_rom*.

O bloco controle gera os sinais necessários para operação dos demais blocos. Foram implementadas uma máquina de estados e contadores específicos para o controle da sobreposição das janelas curtas, que geram a sinalização de amostra válida na saída e sinais de controle dos multiplexadores.

A arquitetura implementa uma máquina de estados que controla a existência de dado válido no registrador de saída do multiplicador. No caso de um bloco de áudio composto por uma janela longa (*only long*, *long start* e *long stop*) o sinal gerado por essa máquina é usado como a saída *dout_ready_o* pois os dados estão prontos para o próximo módulo do decodificador após a multiplicação pelos coeficientes de janela. Quando o bloco é constituído por janelas curtas o sinal gerado pela máquina de estados é usado como controle para a sobreposição das janelas do bloco.

Quatro módulos componentes do bloco de controle geram os sinais de acesso às

tabelas de coeficientes de janela, cada um para um dos valores de *window sequence*. Cada um desses módulos recebe um sinal com o parâmetro *window shape* atual, outro com o *window shape* do bloco anterior e a posição no bloco de áudio da amostra sendo processada. A partir desses sinais é gerado o controle para acesso ao módulo win_rom, onde se encontram as tabelas de coeficientes de janela.

Conforme explicado na seção 4.6, existem 4 possíveis valores para *window sequence* e cada um deles resulta em uma janela com arranjo diferente. O bloco tipo *only long sequence* requer o processamento mais simples: a primeira metade é multiplicada por coeficientes de janela longa usando a função do tipo definido para o bloco anterior e a segunda metade é multiplicada por coeficientes de janela longa do tipo definido pelo *window shape* do bloco atual.

5.3.3. Sobreposição dos blocos

Como a sobreposição necessita armazenar de dados de um *frame* para usar no seguinte, uma área de memória é necessária para cada canal de áudio. Cada *frame* possui 2048 amostras, que devido à sobreposição de 50% das janelas da IMDCT geram 1024 amostras de áudio na saída. Sendo assim, o módulo necessita de 1024 posições de memória com palavras de 16 bits para cada canal.

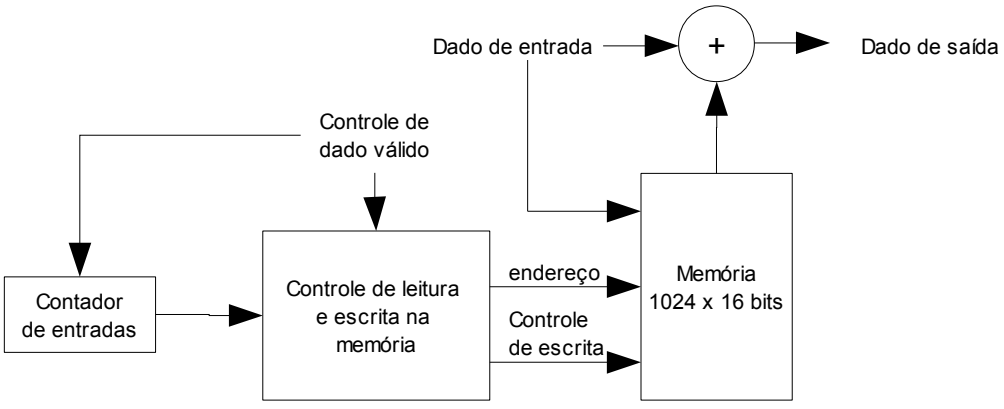


Figura 52: Diagrama de blocos da sobreposição

A operação do módulo é simples, necessita somente de um somador para gerar a amostra de áudio. Como os dados são recebidos na ordem correta, os primeiros 1024 valores recebidos são somados aos correspondentes em memória e enviados para a saída. Os últimos

1024 valores da janela substituem os anteriores na memória para uso no *frame* seguinte. A cada troca de canal a memória em uso é chaveada. A arquitetura proposta possui um somador para gerar a saída, um bloco de controle de leitura e escrita na memória acionado por um contador de entradas.

6. IMPLEMENTAÇÃO DA ARQUITETURA

A arquitetura proposta do decodificador AAC foi implementada em linguagem VHDL para uso em um FPGA. A placa de prototipagem no decorrer do trabalho foi uma Altera DE2. Essa placa possui um FPGA Altera da família Cyclone II com aproximadamente 35 mil elementos lógicos. Além disso, ela possui um *codec* de áudio com 2 canais, usado para reproduzir o áudio do decodificador e uma memória flash de 4 Mbytes usada para armazenar o *bitstream* de áudio codificado.

O processo de verificação dos módulos durante o desenvolvimento foi realizado com o auxilio do *software* de simulação Modelsim. Foi utilizado um *software* em linguagem C desenvolvido por um grupo da UnB.

O decodificador completo com um bloco de comunicação com o *codec* e outro para leitura da memória *flash* ocupou ao todo 26549 elementos lógicos do FPGA, o que equivale a 80% da capacidade do mesmo. A memória total utilizada foi de 248704 bits, 51% da memória disponível no FPGA.

Tabela 4: Consumo de elementos do FPGA por módulo

Módulo	Elementos Lógicos	Memória (em bits)
Bitstream Buffer	120	0
Bitstream Decoder	935	16256
Scale Factor Data	412	4608
Spectral Data	6396	14336
SpecCoefBuffer	5	27648
IMDCT	8936	88064
Window	9040	0
Overlap	190	32768

A estratégia adotada para verificação do funcionamento em placa foi realizar a integração começando pelo módulo de saída para o *codec* pois é muito difícil verificar o resultado sem uma saída sonora. A partir do *codec* a integração dos módulos prossegue no fluxo contrário dos dados, até a integração do último módulo na entrada do *bitstream*.

6.1. Decodificação do Espectro

O bloco de decodificação do espectro na sua maior parte não introduz erro no processo. As únicas operações que podem causar algum desvio do valor esperado quando o decodificador se encontra em operação normal são a quantização inversa e a aplicação dos fatores de escala. A figura 53 mostra a comparação do resultado da decodificação do espectro de um *frame* de janela longa usando os decodificadores em *software* e *hardware*.

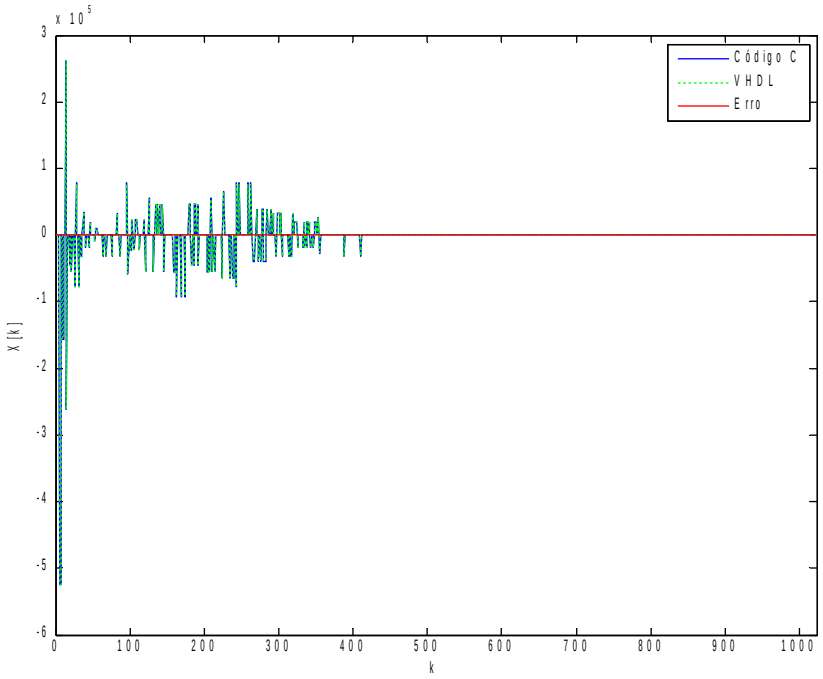


Figura 53: Espectro de janela longa

Para reconstruir o espectro do canal usado no exemplo foram necessários 24564 ciclos, mas, como mencionado anteriormente, esse valor deve variar com os *frames*.

6.2. Decodificação Huffman

Os primeiros módulos a serem desenvolvidos foram para a decodificação Huffman dos fatores de escala e coeficientes espectrais. A verificação de ambos foi feita usando o Modelsim e comparando a saída com os resultados obtidos no *software* de referência. Os ciclos necessários para a execução de cada um dos módulos depende do *bitstream* pois eles fazem a leitura de 1 bit por ciclo e os códigos Huffman tem tamanho variável.

No *frame* usado como exemplo temos dois canais, as janelas são do tipo *Only Long Sequence* e a taxa de amostragem do sinal é de 48 kHz. Para o primeiro canal a decodificação

dos fatores de escala tomou 1206 ciclos e para os coeficientes espectrais foram necessários 22017 ciclos.

6.3. Quantização Inversa e Aplicação de Fatores de Escala

O módulo de quantização inversa pode introduzir erro no processo pois na sua implementação foi usada uma interpolação para diminuir a tabela de valores necessária. Além disso, devido ao formato em ponto fixo da saída, pode haver perda de precisão. No caso da aplicação dos ganhos relacionados aos fatores de escala também pode haver a introdução de mais uma pequena parcela de erro. Em condições normais de operação esses

6.4. IMDCT

Os módulos da IMDCT foram inicialmente verificados de forma separada, para então serem integrados e testados como um módulo único. O objetivo era gerar um módulo capaz de operar com o sinal de relógio a uma frequência de 50 MHz aproveitando um dos sinais disponíveis na placa de prototipagem usada.

6.4.1. IFFT

A arquitetura do módulo IFFT foi desenvolvida (conforme seção 4.3.1.2) para tratar vetores de entrada com 512 pontos. Cada vetor de entrada é composto de 8 blocos de 64 pontos ou um único de 512, dependendo da configuração de janela do *frame*. O módulo IFFT apresenta desempenho diferente para o cálculo de uma IFFT de 512 pontos ou 8 de 64 pontos, conforme os dados na tabela. Isso se deve aos estágios adicionais de rotação dos dados necessários para a IFFT de 512 pontos.

Tabela 5: Características do módulo IFFT

Tipo	Ciclos	Tempo para relógio de 50 MHz
64x8	14265	285,3 us
512x1	20434	408,68 us

A síntese do módulo para o FPGA Cyclone II da Altera usando o ambiente Quartus II consumiu 3079 elementos lógicos, um multiplicador e 32768 bits de área de memória (o

multiplicador e a memória são compartilhados com outros módulos da IMDCT).

Alguns vetores de estímulo usados na verificação do módulo foram gerados aleatoriamente no Matlab e outros foram extraídos de arquivos de áudio codificados em AAC usando o *software* desenvolvido pelo grupo da UnB com algumas modificações.

A verificação do funcionamento foi realizada através de um *testbench* criado em VHDL para uso com o Modelsim, onde os dados de estímulo eram fornecidos ao módulo a partir de um arquivo texto e as saídas eram gravadas em outro arquivo texto. O arquivo com valores gerado pelo *testbench* foi comparado com a saída esperada no Matlab para a verificação do erro causado pelo uso de ponto fixo na execução do cálculo.

Dois exemplos da comparação estão representados nas figuras 54 e 55. Na figura 54 o vetor de entrada foi retirado de uma música codificada em AAC e foi usada uma única janela do bloco *Eight Short Sequence*.

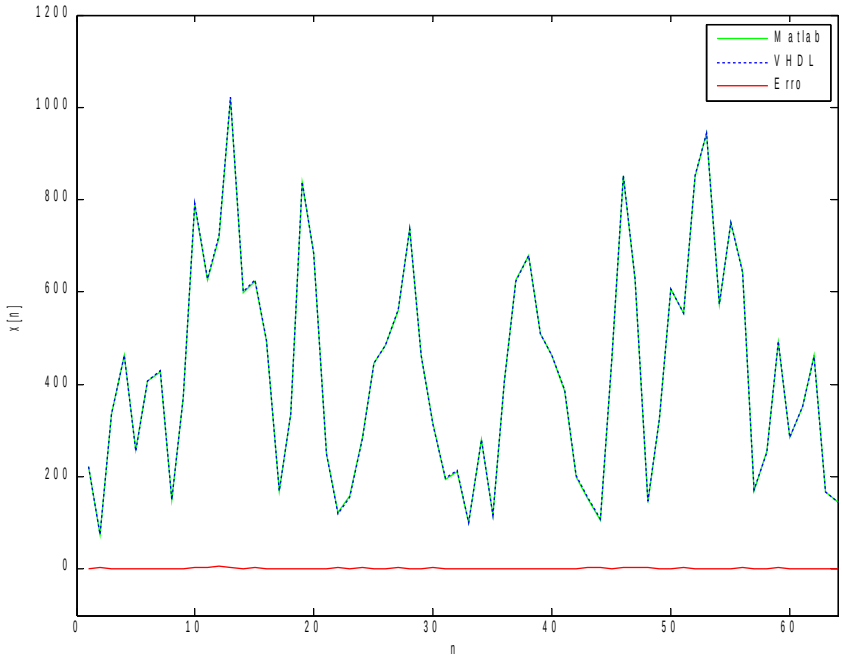


Figura 54: Verificação da IFFT de 64 pontos

A figura 55 mostra a comparação dos resultados da IFFT calculada a partir de um vetor gerado aleatoriamente em Matlab com 512 pontos.

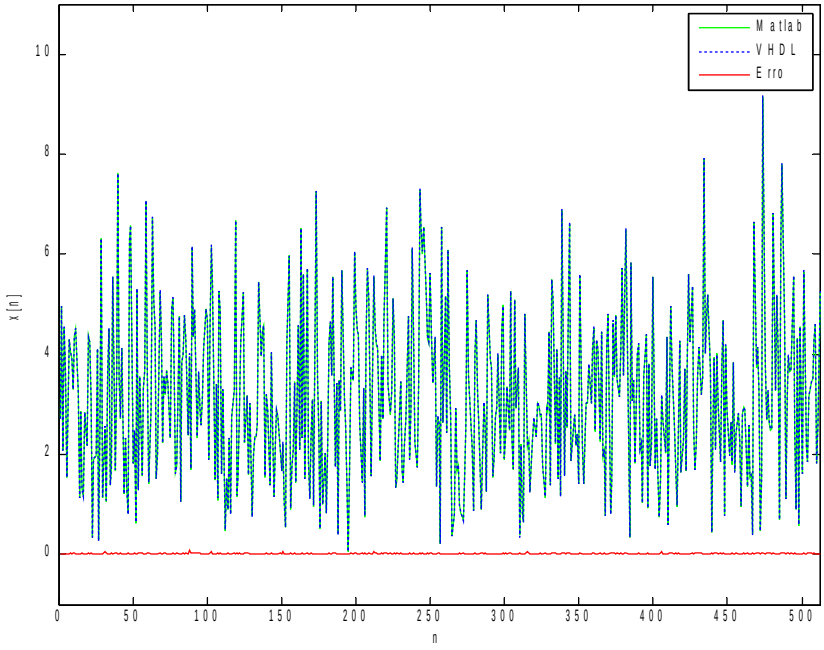


Figura 55: Verificação da IFFT de 512 pontos

6.4.2. JANELAMENTO

Os vetores de estímulo foram gerados com o auxílio de um *software* em linguagem C. O *software* foi usado para gravar arquivos com dados de entrada do bloco de janelamento extraídos de uma música codificada no padrão AAC LC. Foram escolhidos *frames* contendo janelas do tipo *Eight Short Sequence*, *Only Long Sequence*, *Long Start Sequence* e *Long Stop Sequence* para testar os diferentes casos de janelamento. Os dados de referência para comparação com o resultado do módulo em VHDL também foram gravados usando o programa em C. O módulo foi estimulado através de um *testbench* e os valores de saída foram gravados em um arquivo texto para verificação em Matlab.

A figura 56 mostra a comparação da saída do módulo com o resultado obtido através do decodificador em *software*. Apesar do decodificador em C usar dados tipo *float* e o módulo em VHDL utilizar aritmética de ponto fixo, o erro introduzido é pequeno, sendo mais perceptível nos pontos em que a saída do módulo em *hardware* satura devido ao uso de 16 bits para representar a parte inteira dos valores.

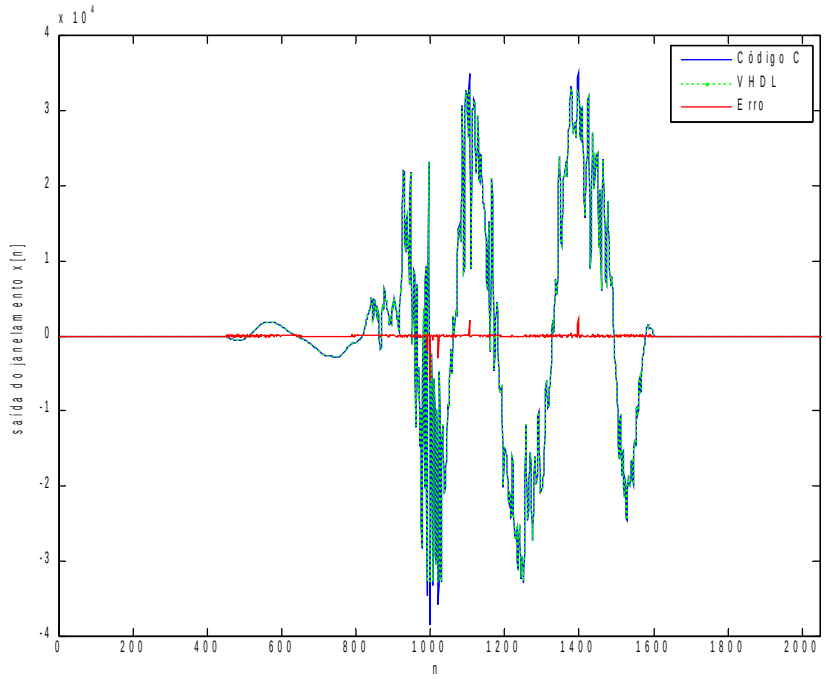


Figura 56: Verificação janelamento Eight Short

Na figura 57 temos a comparação dos resultados obtidos para um *frame* do tipo *Only Long Sequence*.

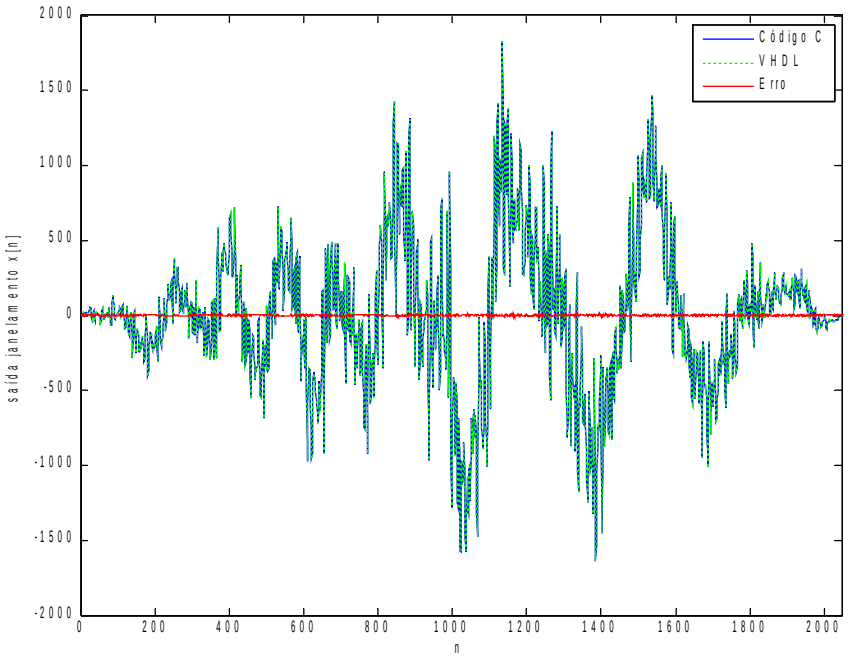


Figura 57: Verificação janelamento Only Long

A figura 58 apresenta a simulação funcional do janelamento para uma janela tipo *Only Long Sequence*. A primeira saída tem uma latência de 3 ciclos e depois disso temos um valor por ciclo, desde que a atualização da entrada se mantenha na mesma velocidade. Na figura o cursor em amarelo marca a primeira entrada válida.

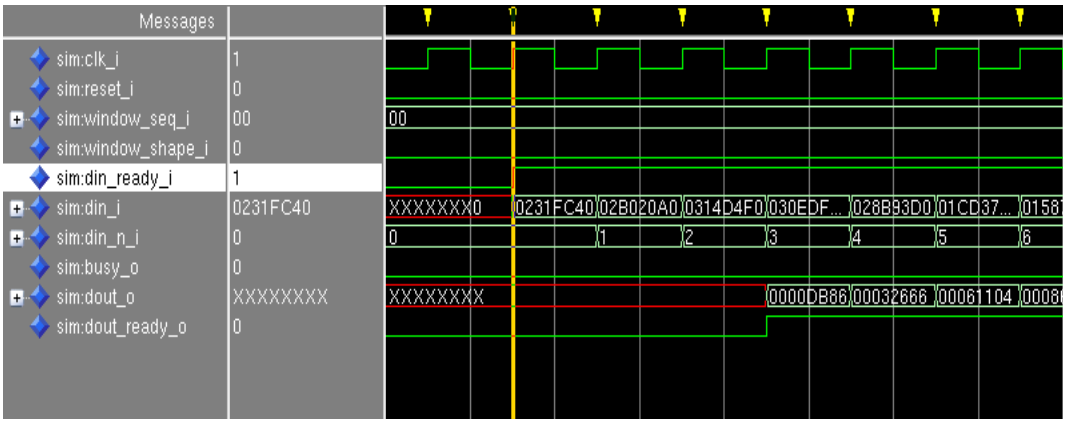


Figura 58: Simulação funcional do janelamento

7. CONCLUSÕES

Os métodos de codificação de áudio se encontram cada dia mais disseminados entre os aparelhos de uso cotidiano. Com a popularização do SBTVD e a futura migração do sistema analógico para o digital a demanda por decodificadores de áudio compatíveis com o sistema só tende a aumentar, portanto o trabalho realizado pode ser de grande valia para a indústria brasileira, servindo de base para um decodificador com suporte total ao SBTVD.

A arquitetura desenvolvida se provou eficaz para a decodificação de *streams* de áudio no formato AAC LC. As ferramentas do padrão MPEG-4 AAC implementadas foram: *bitstream payload deformatter*, *Huffman decoding*, *inverse quantization*, *rescaling* e *block switching / filterbank*. O decodificador suporta a reprodução de até dois canais e taxas de amostragem de 32 kHz, 44.1 kHz e 48 kHz. O circuito sintetizado do decodificador de áudio é capaz de reproduzir áudio estéreo com frequência de amostragem de 48 kHz a partir de um relógio de 4 MHz. A implementação da arquitetura proposta em VHDL sem fazer uso de módulos IPs (propriedade intelectual de terceiros) possibilita que o decodificador seja integrado ao projeto SoC SBTVD sem alterações, onde o objetivo é criar um *System-on-Chip* para a decodificação de áudio e vídeo no padrão SBTVD.

Para a realização desse trabalho o sistema de áudio do SBTVD foi estudado e em consequência o MPEG-4 AAC. As restrições impostas pelo SBTVD em relação ao sistema de codificação de áudio foram utilizadas para reduzir a complexidade da implementação. O texto resultante desse estudo oferece uma visão de todo processo de decodificação de um *stream* AAC LC, analisando em detalhes cada ferramenta do padrão implementada no decodificador. Além disso, a partir do que foi relatado é possível se compreender o funcionamento do algoritmo do ponto de vista teórico, tomando conhecimento dos conceitos psicoacústicos que fundamentam o processo de codificação.

O desenvolvimento de um sistema com esse grau de complexidade é possível graças ao uso de uma linguagem de descrição de *hardware*. A implementação do sistema em VHDL simplifica muito o trabalho do desenvolvedor oferecendo a possibilidade de verificar o funcionamento do circuito a partir de ferramentas de simulação.

Como trabalhos futuros é sugerida a implementação das ferramentas opcionais, como *perceptual noise substitution* e *temporal noise shaping*, que não foram incluídas no trabalho atual e a adição do suporte ao formato 5.1 de áudio. A implementação das ferramentas

spectral band replication e *parametric stereo* é necessária para tornar o decodificador compatível com *streams* HE-AAC versão 1 e 2, abrangendo a totalidade da especificação de áudio do SBTVD. Por fim ainda seria interessante o estudar formas de adaptar o algoritmo para gerar um circuito sintetizado mais eficiente e reduzir o consumo de energia.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR-15602-2**: televisão digital - codificação de vídeo, áudio e multiplexação - parte 2: codificação de áudio. Rio de Janeiro, 2008.

BANG, K. H. et al. Design optimization of MPEG-2 AAC decoder. **IEEE Transactions on Consumer Electronics**, New York, v. 47, n. 4, p. 895-903, Nov. 2001.

BOSI, M. Filter banks in perceptual audio coding. In: AES INTERNATIONAL CONFERENCE: HIGH QUALITY AUDIO CODING, 17., 1999, Florence. **Proceedings...** [S.l.]: AES, 1999. p. 125-133.

BRANDENBURG, K. Introduction to perceptual audio coding. In: GILCHRIST, N.; GREWIN, C. **Collected papers on digital audio bit-rate reduction.** New York: AES, 1996. p. 23-30.

_____. MP3 and AAC explained. In: AES INTERNATIONAL CONFERENCE: HIGH-QUALITY AUDIO CODING, 17., 1999, Florence. **Proceedings...** [S.l.]: AES, 1999. p. 99-110.

BRANDENBURG, K.; BOSI, M. Overview of MPEG audio: current and future standards for low bit-rate audio coding. **AES Journal**, New York, v. 45, n. 1, p. 4-21, 1997a.

_____. ISO/IEC MPEG-2 advanced audio coding: overview and applications. In: AES CONVENTION, 103., 1997, New York. **Proceedings...** [S.l]: AES, 1997b.

BRANDENBURG, K.; KUNZ, O.; SUGIYAMA, A. MPEG-4 natural audio coding. **Signal Processing:** image communication, Amsterdam, v. 15, p. 423-444, Jan. 2000.

CHIANG, W.; HWANG, C.; HSU, Y. Advances in low bit-rate audio coding: a digest of selected papers from recent AES conventions. **AES Journal**, New York, v. 51, n. 10, p. 956-964, 2003.

DIETZ, M. et al. Spectral Band Replication, a novel approach in audio coding. In: AES CONVENTION, 112., 2002, Munich. **Proceedings...** [S.l.]: AES, 2002.

DU, F. et al. An implementation of filterbank for MPEG-2 AAC on FPGA. In: INTERNATIONAL CONFERENCE ON ANTI-COUNTERFEITING, SECURITY AND IDENTIFICATION, 2., 2008, Guiyang. **Proceedings...** [S.l.]: IEEE, 2008, p. 391-394.

DUHAMEL, P.; MAHIEUX, Y.; PETIT, J.P. A fast algorithm for the implementation of filter banks based on “time domain aliasing cancellation”. In: INTERNATIONAL CONFERENCE

ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, 1991, Toronto. **Proceedings...** [S.l.]: IEEE, 1991. p. 2209-2212.

EHRET, A. et al. AacPlus, only a low-bitrate codec. In: AES CONVENTION, 117., 2004, San Francisco. **Proceedings...** [S.l.]: AES, 2004, p. 1-8.

GRILL, B. The MPEG-4 general audio coder. In: AES INTERNATIONAL CONFERENCE: HIGH-QUALITY AUDIO CODING, 17., 1999, Florence. **Proceedings...** [S.l]: AES, 1999, p. 147-156.

HARTMANN, W. M. **Signals, sound and sensation**. New York: Springer-Verlag, 1998. 663 p.

HEE, H. J.; SUNWOO, M. H.; MOON, J. H. Novel non-linear inverse quantization algorithm and its architecture for digital audio codecs. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 2007, New Orleans. **Proceedings...** [S.l.]: IEEE, 2007, p. 357-360.

HERRE, J. Temporal noise shaping, quantization and coding methods in perceptual audio coding: a tutorial introduction. In: AES INTERNATIONAL CONFERENCE: HIGH-QUALITY AUDIO CODING, 17., 1999, Florence. **Proceedings...** [S.l.]: AES, 1999, p. 312-325.

HERRE, J.; JOHNSTON, J. D. Enhancing the performance of perceptual audio coders by using temporal noise shaping (TNS). In: AES CONVENTION, 101., 1996, Los Angeles. **Proceedings...** [S.l.]: AES, 1996.

HOFFMANN, G. A. **Study of the audio coding algorithm of the MPEG-4 AAC standard and comparison among implementations of modules of the algorithm**. 2002. 115 p. Dissertação (Mestrado em Ciências da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

IBGE. **Pesquisa nacional por amostra de domicílios**: síntese de indicadores. 2009. Disponível em: <http://www.ibge.gov.br/home/estatistica/populacao/trabalhoerendimento/pnad2009/tabelas_pdf/sintese_ind_6_4.pdf>. Acesso em: 10 jun. 2011.

ISO/IEC. **ISO/IEC 13818-7**: information technology – generic coding of moving pictures and associated audio information – part 7: advanced audio coding (AAC). Zurich, 2006.

_____. **ISO/IEC 14496-3**: information technology – coding of audio-visual objects – part 3: audio. Zurich, 2005.

KUNZ, O.; BRANDENBURG, K. An overview of MPEG-4 audio. In: AES UK CONFERENCE: AUDIO – THE SECOND CENTURY, 14., 1999, London. **Proceedings...** [S.l.]: AES, 1999, p. 137-145.

LI, L. et al. Efficient architectures of MDCT/IMDCT implementation for MPEG audio codec. In: INTERNATIONAL CONFERENCE ON ANTI-COUNTERFEITING, SECURITY AND IDENTIFICATION IN COMMUNICATION, 3., 2009, Hong Kong. **Proceedings …** 2009, p. 156-159.

LINNEMAN, R. **Advanced audio coding on an FPGA**. 2002. 126 p. Dissertação (Bacharel em Engenharia Elétrica) – School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, 2002. Disponível em: <http://www.mp3-tech.org/programmer/docs/linneman_thesis.pdf>. Acesso em: 12 mar. 2009

LIU, C. et al. Design of MPEG-4 AAC encoder. In: AES CONVENTION, 117., 2004, San Francisco. **Proceedings...** [S.l.]: AES, 2004.

_____. Compression artifacts in perceptual audio coding. In: AES CONVENTION, 121., 2006, San Francisco. **Proceedings... .** [S.l.]: AES, 2006.

MOORE, B. Masking in the human auditory system. In: GILCHRIST, N.; GREWIN, C. **Collected Papers on Digital Audio Bit-Rate Reduction**. New York: AES, 1996. p. 9-19.

OPPENHEIM, A. V.; SCHAFER, R. W. **Discrete-time signal processing**. Upper Saddle River: Prentice-Hall, 1998.

PRINCEN, J. P.; BRADLEY, A. B. Analysis/synthesis filter bank design based on time domain aliasing cancellation. **IEEE Transactions on Acoustics, Speech and Signal Processing**, [S.l.], v. 34, n. 5, p. 1153-1161, Oct. 1986.

PURNHAGEN, H. An overview of MPEG-4 audio version 2. In: AES INTERNATIONAL CONFERENCE: HIGH-QUALITY AUDIO CODING, 17., 1999, Florence. **Proceedings...** [S.l.]: AES, 1999, p. 157-168.

SHENOY, R. R.; NAIK, S. S.; SUMAM, D. S. Improved algorithms for implementation of MPEG2 AAC decoder on FPGA. In: IEEE REGION 10 TENCON, 2005, Melbourne. **Proceedings...** [S.l.]: IEEE, 2005, p. 1-4.

SPANIAS, A.; PAINTER, T.; ATTI, V. **Audio signal processing and coding**. New Jersey: Wiley, 2007. 464 p.

TSAI, T.; LIU, C. Low-power system design for MPEG-2/4 AAC audio decoder using pure ASIC approach. **IEEE Transactions on Circuits and Systems**, [S.l.], v. 56, n. 1, p. 144-155, Jan. 2009.

TSAI, T.; LIU, J. Architecture design for MPEG-2 AAC filterbank decoder using modified regressive method. In: INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 2002, Orlando. **Proceedings...** [S.l.]: IEEE, 2002a, p. 3216-3219.

TSAI, T.; YEN, C. A high quality re-quantization/quantization method for MP3 and MPEG-4 AAC audio coding. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 2002, Phoenix. **Proceedings...** [S.l.]: IEEE, 2002b, p. 851-854.

TSAI, T. et al. A 1.4 MHz 0.21 mW MPEG-2/4 AAC single chip decoder. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, 31., 2009, San Jose. **Proceedings...** [S.l.]: IEEE, 2009, p. 653-656.

WOLTERS, M. et al. A closer look into MPEG-4 high efficiency AAC. In: AES CONVENTION, 115., 2003, . **Proceedings...** [S.l.]: AES, 2003, p. 1-16.

ZHANG, H.; LU, M.; WANG, G. An ASIC implementation of MPEG audio decoders. In: INTERNATIONAL CONFERENCE ON ASIC, 7., 2007, Guilin. **Proceedings...** [S.l.]: IEEE, 2007, p. 754-757.