UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RONALDO HÜSEMANN

ARQUITETURA DE CO-PROJETO HARDWARE/SOFTWARE PARA IMPLEMENTAÇÃO DE UM CODIFICADOR DE VÍDEO ESCALÁVEL PADRÃO H.264/SVC

Porto Alegre
2011

RONALDO HÜSEMANN

ARQUITETURA DE CO-PROJETO HARDWARE/SOFTWARE PARA IMPLEMENTAÇÃO DE UM CODIFICADOR DE VÍDEO ESCALÁVEL PADRÃO H.264/SVC

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica (PPGEE), da Universidade Federal do Rio Grande do Sul (UFRGS), como parte dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica.
Área de concentração: Engenharia de Computação: Processamento de Sinais

ORIENTADOR: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre

2011

RONALDO HÜSEMANN

ARQUITETURA DE CO-PROJETO HARDWARE/SOFTWARE PARA IMPLEMENTAÇÃO DE UM CODIFICADOR DE VÍDEO ESCALÁVEL PADRÃO H.264/SVC

Este documento foi julgado adequado como tese do aluno para o Programa de Pós-Graduação em Engenharia Elétrica da UFRGS.

Orientador: _____

Prof. Dr. Altamiro Amadeu Susin, UFRGS

Doutor em Informática, INPG - Grenoble, França

Banca Examinadora:

Prof. Dr. Sérgio Bampi - PPGC-UFRGS

Prof. Dr. Ivan Saraiva Silva - Depto. Informática e Estatística - UFPI;

Prof. Dr. César Albenes Zeferino - CTTMAR-UNIVALI;

Prof. Dr. Marcelo Soares Lubaszewski - PPGEE-UFRGS;

Prof. Dr. Gilson Inácio Wirth - PPGEE-UFRGS

Coordenador do PPGEE: _____

Prof. Dr. Alexandre Sanfelice Bazanella

Porto Alegre, agosto de 2011.

Dedico este trabalho a meu pai, Erli, a minha mãe, Nelcy, pelo incondicional apoio, em todos os momentos, e a meu irmão Rudimar por sua grande consideração.

AGRADECIMENTOS

Ao final deste trabalho, não seria justo deixar de agradecer a várias pessoas que me ajudaram no desenvolvimento deste projeto.

Em especial destaque vem os colegas de laboratório e pesquisa Ricardo Kintschner, Alfredo Capella, Mariano Majolo, Ricieri Clayton da Silva, Eduardo Eick, Daniel dos Santos, Victor Guimarães, Maurício Dau, Davi Fagundes, Daniel Weber, Anderson Giacomolli, Augusto Lenz, Diego Schwingel, entre outros. Percebe-se como um trabalho de grande porte somente pode ser executado com êxito devido ao apoio de uma grande equipe.

Um agradecimento também aos colegas professores Robson Schaeffer, Rodrigo Porto e Marcelo Malheiros, que mesmo não participando diretamente desta pesquisa, sempre torceram muito para que tudo corresse bem.

Por fim, vale uma menção aos professores da UFRGS e coordenadores de projeto Altamiro Susin, Valter Roesler e José Valdeni, pela confiança e liberdade cedidas na condução desta tão relevante pesquisa.

RESUMO

Visando atuação flexível em redes heterogêneas, modernos sistemas multimídia podem adotar o conceito da codificação escalável, onde o fluxo de vídeo é composto por múltiplas camadas, cada qual complementando e aprimorando gradualmente as características de exibição, de forma adaptativa às capacidades de cada receptor. Atualmente, a especificação H.264/SVC representa o estado da arte da área, por sua eficiência de codificação aprimorada, porém demanda recursos computacionais extremamente elevados. Neste contexto, o presente trabalho apresenta uma arquitetura de projeto colaborativo de hardware e software, que explora as características dos diversos algoritmos internos do codificador H.264/SVC, buscando um adequado balanceamento entre as duas tecnologias (hardware e software) para a implementação prática de um codificador escalável de até 16 camadas em formato de 1920x1080 pixels. A partir de um modelo do código de referência H.264/SVC, refinado para reduzir tempos de codificação, foram definidas estratégias de particionamento de módulos e integração entre entidades de software e hardware, avaliando-se questões como dependência de dados e potencial de paralelismo dos algoritmos, assim como restrições práticas das interfaces de comunicação e acessos à memória. Em hardware foram implementados módulos de transformadas, quantização, filtro anti-blocagem e predição entre camadas, permanecendo em software funções de gerência do sistema, entropia, controle de taxa e interface com usuário. A solução completa obtida, integrando módulos em hardware, sintetizados em uma placa de desenvolvimento, com o software de referência refinado, comprova a validade da proposta, pelos significativos ganhos de desempenho registrados, mostrando-se como uma solução adequada para aplicações que exijam codificação escalável tempo real.

Palavras-chave: Codificação de vídeo escalável. Padrão H.264/SVC. Co-projeto de hardware e software. Lógicas programáveis.

ABSTRACT

In order to support heterogeneous networks and distinct devices simultaneously, modern multimedia systems can adopt the scalability concept, when the video stream is composed by multiple layers, each one being responsible for gradually enhance the video exhibition quality, according to specific receiver capabilities. Currently the H.264/SVC specification can be considered the state-of-art in this area, by improving the coding efficiency, but, in the other hand, impacting in extremely high computational demands. Based on that, this work presents a hardware/software co-design architecture, which explores the characteristics of H.264/SVC internal algorithms, aiming the right balancing between both technologies (hardware and software) in order to generate a practical scalable encoder implementation, able to process up to 16 layers in 1920x1080 pixels format. Based in an H.264/SVC reference code model, which was refined in order to reduce global encoding time, the approaches for module partitioning and data integration between hardware and software were defined. The proposed methodology took into account characteristics like data dependency and inherent possibility of parallelism, as well practical restrictions like influence of communication interfaces and memory accesses. Particularly, the modules of transforms, quantization, deblocking and inter-layer prediction were implemented in hardware, while the functions of system management, entropy, rate control and user interface were kept in software. The whole solution, which was obtained integrating hardware modules, synthesized in a development board, with the refined H.264/SVC reference code, validates the proposal, by the significant performance gains registered, indicating it as an adequate solution for applications which require real-time video scalable coding.

Keywords: Scalable video coding. H.264/SVC standard. Hardware/software co-design. Programmable logics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ASIC : Application Specific Integrated Circuit

AVC : Advanced Video Coding

B : Bidirectionally-predictive

BS : Boundary Strength

CABAC: Context Adaptive Binary Arithmetic Coding

CAVLC: Context Adaptive Variable Lenght Coding

CGS : Coarse Grain Scalability

CIF : Common Intermediate Format

COP : Column of Pixels

CPU : Central Processing Unit

DCT : Discrete Cosine Transform

DMA : Direct Memory Access

DS : Diamond Search

DSP : Digital Signal Processing

EPZS : Enhanced Predictive Zonal Search

ESS : Extended Spatial Scalability

FGS : Fine Grain Scalability

FIFO : First Input First Output

FPGA : Field Programmable Gate Array

FSS : Four Step Search

GOP : Group of Pictures

GPGPU: General Purpose Graphical Processing Unit

GPU : Graphical Processing Unit

HD : High Definition

I : Intraframe

IEC : International Electrotechnical Commission

ILP : Inter-Layer Prediction

IP : Internet Protocol

IPTV : Internet Protocol TeleVision

ISO : International Standardization Organization

ITU : International Telecommunication Union

JSVM : Joint Scalable Video Model

JVT : Joint Video Team

LOP : Line of Pixels

LSB : Least Significant Bit

LUT : Look-Up Table

MCTF : Motion Compensated Temporal Filtering

MGS : Medium Grain Scalability

MPEG : Moving Picture Expert Group

MSB : Most Significant Bit

MSE : Mean Squared Error

NAL : Network Abstraction Layer

NOC : Network Of Chip

NTSS : New Three-Step Search

P : Predictive

PMVFAST : Predictive Motion Vector Field Adaptive Search Technique

PSNR : Peak Signal Noise Ration

QOS : Quality Of Service

QP : Quantization Parameter

RAM : Random Access Memory

ROM : Read Only Memory

SAD : Sum of Absolute Differences

SBTVD: Sistema Brasileiro de TV Digital

SD : Standard Definition

SIMD : Single Instruction Multiple Data

SNR : Signal to Noise Ratio

SOC : System On Chip

SRAM: Static Random Access Memory

SVC : Scalable Video Coding

TIPS : Tera Instructions Per Second

TLP : Transaction Layer Packet

TSS : Three-Step Search

TSB : Temporal Sub Band

VHDL : VHSIC Hardware Description Language

VHSIC: Very High Scale Integrated Circuit

VLC : Variable Length Coding

VLIW : Very-Large Instruction Word

VOL : Video Object Layer

SUMÁRIO

1 INTRODUÇÃO

A evolução contínua das aplicações de multimídia tem modificado a realidade da sociedade ano a ano. Aplicações comerciais consideradas inviáveis ou até mesmo utópicas algumas décadas atrás, tais como televisão de alta definição, videofone no celular, telereunião, ensino a distância com interação por voz e imagem, consultas por telemedicina, entre outras, já estão em funcionamento atualmente devido à adoção das tecnologias de áudio e vídeo digitais (KANT; MITHUN; GUPTA, 2006).

Uma das características das tecnologias de áudio e vídeo digitais está na possibilidade de alta compressão de dados, a fim de reduzir as bandas dos canais de comunicação ou espaços de armazenamento ocupados, facilitando assim sua implementação prática. Os conjuntos de algoritmos responsáveis por esta compressão, são respectivamente chamadas de codificadores de áudio e vídeo. Considerando-se especificamente os codificadores de vídeos, que trabalham com os maiores volumes de dados, podem-se citar vários padrões internacionais, tais como MPEG-2, H.263, WMV, MPEG-4, H.264/AVC, entre outros (KEITH, 2004).

De uma forma geral, pode-se dizer que estes codificadores, apesar de serem padrões distintos, apresentam uma arquitetura similar, fortemente baseada no modelo definido pelas normas da entidade –MPEG (*Moving Picture Expert Group*), que explora a remoção da redundância espacial de dados, a partir do emprego de transformadas, quantização e codificação de entropia, e da redundância temporal, a partir de algoritmos de estimativa de movimento (RICHARDSON, 2003).

A evolução dos codificadores de vídeo levou a grandes ganhos no quesito eficiência de compressão. Soluções atuais permitem a manipulação de centenas de milhões de bytes de vídeo por segundo, comprimindo os mesmos por diversas dezenas a centenas de vezes (RIECKL, 2008).

Ainda hoje, entretanto, apesar da alta eficiência de compressão obtida pelos codificadores de vídeo atuais se observam algumas dificuldades destes para operar sobre redes heterogêneas ou híbridas. Por redes heterogêneas entendem-se as redes de composição mista, que interligam, ao mesmo tempo, dispositivos com diferentes características de poder de processamento, resolução de vídeo ou capacidades dos canais de comunicação (MARPE; WIEGAND; HERTZ, 2006).

O maior problema destas redes é que os dispositivos de menor capacidade de processamento ou menor banda de recepção (como, por exemplo, aparelhos celulares, IPODs e dispositivos *tablets*) não conseguem nativamente lidar com vídeos de alta resolução. Assim, ao se utilizar um vídeo codificado em alta resolução estar-se-ia eliminando a presença destes dispositivos da rede.

Por outro lado, restringir o fluxo de vídeo para menores resoluções, por conta destes dispositivos mais limitados, geraria um subaproveitamento de recursos dos dispositivos de maior capacidade (como, por exemplo, computadores de alto desempenho e receptores de televisão digital), o que também não é desejado.

Para atender a este dilema, cada vez mais comum devido à popularização das redes Ethernet, essencialmente heterogêneas, surgiu o conceito da escalabilidade sobre codificadores de vídeo. Escalabilidade de vídeo é o recurso que possibilita armazenar ou transmitir simultaneamente as informações de um dado vídeo em diversas partes, normalmente chamadas de **camadas**, cada qual contendo um nível de detalhamento diferente (OHM, 2005).

A idéia por trás do conceito de escalabilidade é de flexibilizar a produção de fluxos codificados de modo que os diferentes dispositivos consumam apenas as informações que conseguem individualmente processar, sem que isso afete os modos de exibição dos demais dispositivos que compartilham a mesma rede (SCHIERL et al,. 2007).

Utilizando-se o conceito em que o vídeo é transmitido em diferentes camadas, a rede e os dispositivos podem ser ajustados para receber e processar as camadas ótimas relativas às suas capacidades.

Para ilustrar mais claramente esta questão, podem-se citar duas situações principais onde o recurso de escalabilidade tem grande aplicabilidade:

a) Redes compartilhadas por dispositivos com diferentes capacidades;
b) Redes com canais de comunicação distintos ou não confiáveis.

Na primeira situação citada, os diferentes dispositivos que compõem a rede podem obter vídeos com resolução e taxa de atualização dependentes do número de camadas recebidas.

Nesse caso, a quantidade de camadas a serem recebidas e processadas por cada dispositivo deve ser escolhida segundo suas próprias capacidades. Quanto mais camadas forem processadas, maior será o detalhamento do vídeo a ser exibido.

Na Figura 1, pode-se observar um exemplo ilustrativo deste tipo de situação, onde três tipos de dispositivos compartilham uma mesma rede sem fio.

Uma vez que cada dispositivo apresenta uma capacidade de processamento distinta, o vídeo será ajustado para cada um destes atendendo a diferentes resoluções e taxas de amostragem.

Particularmente, no exemplo apresentado, o aparelho celular recebe e processa apenas uma camada escalável, descartando as demais camadas transmitidas (linhas tracejadas), e com isso consegue obter um vídeo de resolução CIF (*Common Intermediate Format*) a uma taxa de 15 quadros por segundo. O computador portátil, por sua vez, processa três camadas escaláveis e com isso consegue obter um vídeo de maior resolução, no caso no formato SD (*Standard Definition*) a uma taxa de 30 quadros por segundo. O dispositivo mais eficiente de todos (televisor) é o único que consegue processar todas as camadas e exibir o vídeo conforme gerado no seu formato (HD - *High Definition*) e taxa de exibição originais.

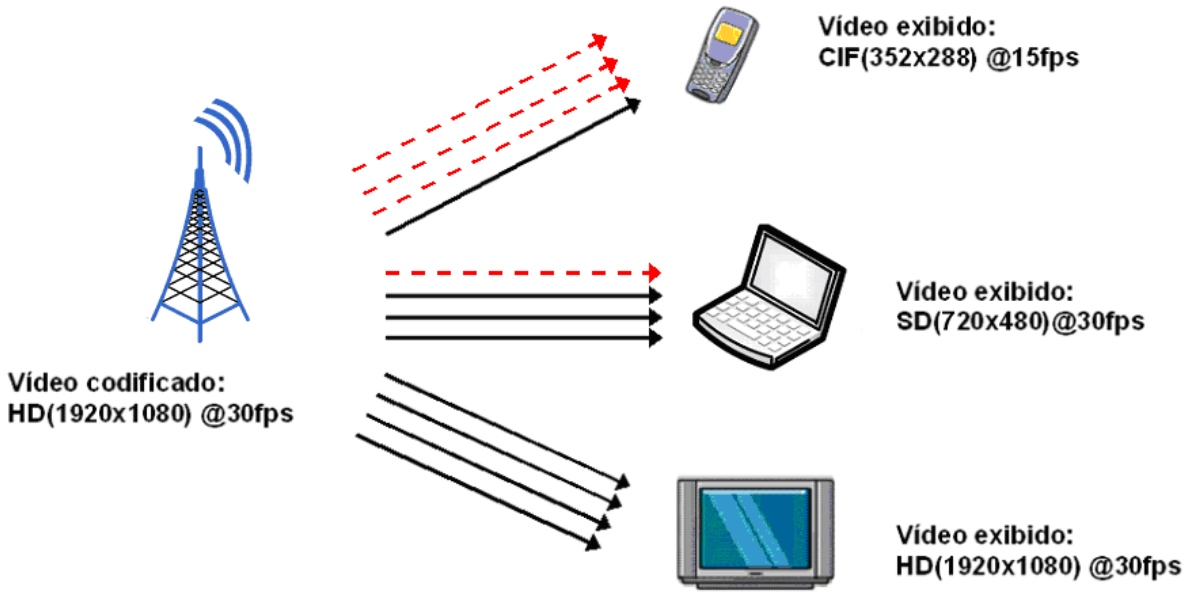


Figura 1: Exemplo de uma aplicação de rede heterogênea.

Na segunda situação típica, são os canais de comunicação que limitam a exibição, fazendo com que alguns dispositivos consigam receber efetivamente apenas parte das camadas geradas pelo codificador. O uso do recurso de escalabilidade nestas aplicações traz a vantagem adicional de tornar o sistema mais robusto à ocorrência da perda de pacotes, que, apesar de pouco desejada, é comum em redes baseadas em Ethernet com ou sem fio (RUHIO-LOYOLA; SALA; ALI, 2008).

Em aplicações que empregam codificação de vídeo escalável a perda de pacotes normalmente compromete apenas uma das camadas. Com as demais camadas recebidas a exibição ainda será possível, mesmo que com alguma eventual redução de qualidade. Esta é uma vantagem interessante, pois em aplicações que utilizam codificação não escalável, a perda de pacotes geralmente causa o desconfortável efeito do surgimento de artefatos de blocagem ou mesmo congelamento do vídeo. Como exemplo desta necessidade pode-se citar as aplicações de IPTV (*Internet Protocol TeleVision*) em um mercado que cresce dezenas de milhões de unidades anualmente (BOGEN, 2010).

A Figura 2 apresenta um exemplo deste tipo de aplicação, onde percebe-se que o computador A possui um canal de menor capacidade (apenas uma camada de vídeo é suportada). Os computadores B e C comportam o mesmo número de camadas, porém, neste exemplo, o computador B perdeu uma das camadas por falha da rede. Devido a esta falha o vídeo processado pelo mesmo acabou perdendo parte das informações (taxa de exibição reduziu), mas continuou sendo processado e exibido, de forma quase transparente para o usuário. O computador C recebeu todas as camadas e por isso exibe o vídeo com as características de resolução e taxa de exibição máximas do sistema.

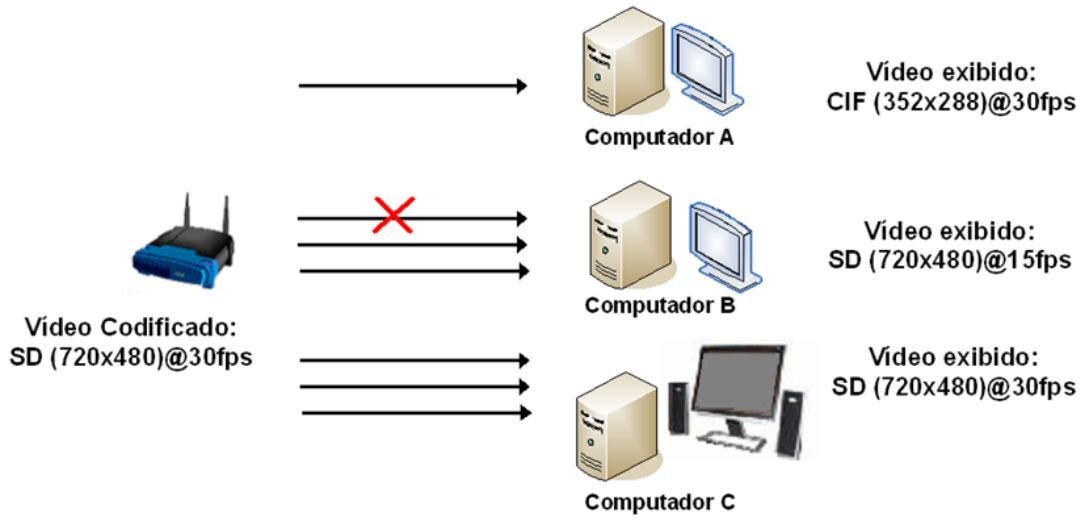


Figura 2: Exemplo de uma aplicação de redes de comunicação distintas ou não confiáveis.

Na realidade, as situações apresentadas não são excludentes. Muitas vezes, as redes de receptores multimídia reais são formadas por um misto das duas situações, ou seja, comportam, ao mesmo tempo, dispositivos com diferentes capacidades sobre canais de comunicação com bandas variáveis e sujeitas a perdas de pacotes. Esta condição mista exige uma flexibilidade ainda maior para os codificadores de vídeo.

Considerando o exposto, mostra-se evidente a importância do conceito de escalabilidade aplicada a codificadores de vídeo quando operando sobre redes heterogêneas, com especial destaque para o padrão escalável H.264/SVC (*Scalable Vídeo Coding*) que é considerado atualmente como estado da arte desta tecnologia (NARVEKAR et al., 2009). A implementação prática de um codificador de vídeo deste, entretanto, é computacionalmente onerosa, devido à elevada complexidade envolvida.

A estrutura excepcionalmente complexa de um codificador escalável se explica, via de regra, por este ser internamente composto por diversos codificadores inter-relacionados (cada um deles associado a uma camada de vídeo distinta). Além disso, deve-se considerar o fato do codificador H.264/SVC, em especial, possuir, em sua estrutura, diversos algoritmos aprimorados para aumentar a taxa de compressão, os quais aumentam ainda mais a carga computacional do codificador.

Baseado neste contexto, a definição de uma solução que consiga atender em tempo-real as complexas demandas exigidas pela especificação H.264/SVC se apresenta ainda atualmente como um grande desafio para a comunidade científica.

Sendo assim, o objetivo do presente trabalho é desenvolver uma arquitetura adequada para a implementação prática e eficiente de um codificador de vídeo padrão H.264/SVC, visando, principalmente, uso em aplicações que exijam adaptação dinâmica a redes heterogêneas.

Para tanto se introduz uma solução inovadora, baseada em um conceito de co-projeto de hardware e software, ou, mais particularmente, de uma arquitetura própria de projeto colaborativo de hardware e software, a qual busca distribuir os diversos algoritmos internos de um codificador H.264/SVC entre as duas tecnologias (módulos em hardware e rotinas em software).

Como plataforma de trabalho e validação o projeto em questão considerou uma arquitetura simples, composta por um computador pessoal (parte de software), onde roda o software de referência do SVC (JOINT SCALABLE TEAM MODEL, 2010) e uma placa de desenvolvimento com lógicas programáveis (parte de hardware).

Neste sentido, diversos módulos dedicados foram desenvolvidos e adaptados para operar de forma colaborativa visando melhorar o desempenho da solução global de codificação escalável.

De forma geral, o trabalho teve como meta gerar uma solução que permita a codificação escalável de até 16 camadas em formato de alta definição (HD - *High Definition*), ou seja devendo suportar o processamento de até 480 quadros em resolução de 1920x1080 pixels por segundo (16 x 30 qps = 480 qps).

A fim de viabilizar o atendimento de demandas tão altas foram realizados vários ensaios práticos sobre o software de referência SVC, onde foram medidos para diferentes configurações, resultados práticos de qualidade, desempenho global (tempo de execução) e demandas por operações computacionais.

Os resultados obtidos experimentalmente permitiram identificar os blocos mais e menos críticos e com isso gerar a especificação de um modelo refinado de codificador que apresenta menores tempos de codificação, sem impactos significativos na qualidade de vídeo gerada.

A partir deste modelo refinado, foi realizado o mapeamento dos diferentes algoritmos internos entre módulos de hardware e software, levando-se em conta questões práticas como a dependência de dados e potencial de paralelismo individual de cada algoritmo, de forma a se produzir soluções que permitissem redução das demandas por recursos computacionais e principalmente aumentos reais do desempenho.

Além disso, para que o mapeamento realizado fosse de fato efetivo, o projeto colaborativo precisou considerar gargalos pouco explorados, como é o caso das interfaces de comunicação e acessos a memória, de forma a contornar ou pelo menos a reduzir o efeito destes no desempenho global da solução.

No modelo colaborativo desenvolvido, a parte em software fica responsável pelas tarefas de gerenciamento, entropia, controle de taxa e interface com o usuário, enquanto que a parte em hardware realiza as tarefas computacionalmente mais complexas, tais como algoritmos de transformadas, quantização e filtragem (codificação intra-quadro), assim como a estimativa de movimento (codificação inter-quadro).

De forma mais particular, a solução de codificação intra-quadro desenvolvida para a implementação das diferentes camadas, adota como estratégia de projeto a execução iterativa sobre módulos computacionais de alto desempenho.

Com isso os módulos de hardware desenvolvidos podem ser reusados na solução, diminuindo seu tempo ocioso e permitindo taxas efetivamente altas de codificação ao mesmo tempo em que se simplifica a área total da solução.

Já para a solução de codificação inter-quadro, adotou-se como base um algoritmo otimizado de predição baseado em pesquisa zonal, o qual se utiliza de informações de blocos vizinhos, que tanto podem ser da mesma camada ou de camadas consecutivas, como ponto de partida para a determinação das correlações temporais entre quadros.

Esta estratégia se apresenta especialmente importante para um codificador H.264/SVC, onde as informações de camadas vizinhas devem, de fato, ser consideradas durante o processamento das camadas de enriquecimento, visando-se reduzir a taxa de dados ocupada, assim como o tempo de codificação.

A solução desenvolvida se introduz como uma relevante contribuição para a comunidade científica por apresentar uma plataforma colaborativa e prática para o desenvolvimento de aplicações de codificação de vídeo escalável, auxiliando pesquisadores que pretendam trabalhar nesta da área.

Além disso, destaca-se que a plataforma de hardware especificada para este projeto, juntamente com as estratégias de particionamento e integração adotadas, podem ser utilizadas como modelo prático para a implementação igualmente eficiente de outros algoritmos complexos tais como codificação multi-vista, criptografia de dados, algoritmos genéticos de reconhecimento padrões (detecção de face e movimentos), entre outros.

Este trabalho está dividido da seguinte forma: o Capítulo 2 traz uma análise do conceito de escalabilidade sobre codificadores de vídeo, apresentando a evolução das principais tecnologias de codificadores de vídeo, com destaque principal para os padrões escaláveis; o Capítulo 3 apresenta, de forma mais detalhada, as técnicas relacionadas mais diretamente com o padrão escalável H.264/SVC; o Capítulo 4 apresenta a metodologia adotada durante o desenvolvimento desta tese, destacando-se as tomadas de decisão do projeto; o Capítulo 5 descreve a estrutura interna e princípio de funcionamento dos diversos módulos de hardware desenvolvidos especificamente para esta solução colaborativa; o Capítulo 6 apresenta resultados experimentais, comparando os resultados obtidos inicialmente cada um dos módulos de hardware desenvolvidos em relação a outros trabalhos relacionados e posteriormente avaliando a solução completa desenvolvida, desta solução quando aplicada sobre diferentes vídeos de referência, destacando assim os ganhos de desempenho finais

registrados, enquanto que o Capítulo 7, por fim, traz as considerações finais sobre o trabalho desenvolvido, assim como propostas de desdobramentos e projetos futuros.

2 CONCEITO DA ESCALABILIDADE

A maioria dos codificadores de vídeo foi desenvolvida visando aplicabilidade em um modelo de difusão global de fluxo único, como é, por exemplo, o caso dos padrões de televisão digital. Neste modelo, cada emissora é detentora de uma específica faixa de freqüência, onde transmite seu sinal codificado simultaneamente para todos os dispositivos receptores. O canal de comunicação utilizado, seja ele do tipo terrestre, por cabo ou satélite, tem a mesma largura de banda para todos os pontos de cobertura. Qualquer equipamento receptor, independentemente de tamanho ou modelo, deve ser plenamente capaz de receber e processar todas as informações enviadas pela emissora (KEITH, 2004).

Este modelo, entretanto, não se adapta bem para aplicações multimídia mais recentes, como, por exemplo, o caso das aplicações que precisam rodar sobre redes IP (*Internet Protocol*), independentemente do uso de meios físicos com (Ethernet) ou sem fio (Wi-Fi ou WiMax). Notoriamente, estas aplicações, visando maior abrangência e flexibilidade, são formadas por redes heterogêneas. Como citado anteriormente, as redes heterogêneas são compostas por dispositivos com diferentes restrições (capacidade de processamento e/ou recursos de exibição), interligados por canais de comunicação híbridos (distintas topologias de conexão com taxas de transmissão variáveis por canal). Além disso, os canais de comunicação baseados em redes IP são inerentemente sujeitos a perdas de pacotes, o que pode representar um grande problema para codificadores convencionais (HILLESTAD et al., 2007). De fato, para atender de maneira flexível a redes heterogêneas, sistemas de multimídia podem promover a difusão da informação multimídia sob várias configurações (diferentes resoluções e/ou taxas de exibição), permitindo flexibilizar o sistema para se adequar a diversos requisitos e restrições.

2.1 PRINCÍPIO BÁSICO DA ESCALABILIDADE

Um dos primeiros métodos utilizados pela comunidade científica para resolver os problemas citados é chamado de *simulcast* (WIEN, SCHWARZ; OELBAUM, 2007). O método *simulcast* propõe o envio simultâneo de dois ou mais fluxos codificados do mesmo vídeo. Cada um destes fluxos possui características diferentes (resolução do vídeo, taxa de amostragem de quadros, entre outras) e, conseqüentemente, possui uma banda de transmissão distinta.

Cada fluxo é autocontido, ou seja, pode ser completamente decodificado sem levar em conta os demais fluxos transmitidos em paralelo. De forma geral, a banda total ocupada pelo este método é dada pela soma das bandas individuais de cada fluxo.

Para a implementação prática do método *simulcast*, normalmente as entidades transmissoras utilizam-se de diversos codificadores de vídeo em paralelo, cada qual configurado para gerar em sua saída um dos fluxos desejados. Como exemplo prático de aplicação comercial deste método, pode-se citar o SBTVD (Sistema Brasileiro de TV Digital). No SBTVD um segmento do canal de freqüências de cada emissora é reservado para a transmissão de um vídeo de menor resolução visando dispositivos móveis (celulares, laptops e *tablets*), enquanto que os demais segmentos compõem um vídeo de alta definição, visando equipamentos residenciais (terminais de acesso ou televisores com decodificador incluso). Apesar de representar uma estratégia relativamente simples de implementação, existem algumas desvantagens da utilização do método de *simulcast*, dentre as quais pode-se destacar o aumento progressivo da banda de transmissão ocupada (dependendo do número de fluxos simultâneos adotados) e a topologia fechada, onde cada fluxo é gerado visando os específicos requisitos dos dispositivos alvos, os quais precisam assim ser definidos no momento do projeto inicial, impedindo que a solução se adapte a mudanças dinâmicas da rede.

Buscando-se uma solução mais adequada, no início da década de 90 passou-se a pesquisar a utilização do recurso de escalabilidade para codificadores de vídeo, visando resolver estes dois problemas, ou seja, buscando, ao mesmo tempo, maior otimização da banda ocupada e adaptabilidade aos dispositivos e meios de comunicação para redes heterogêneas (OHM, 2005).

O recurso de escalabilidade de vídeo, ao contrário do método *simulcast*, não produz diversos fluxos autocontidos, mas sim fluxos de vídeos relacionados e complementares, que procuram explorar a redundância de informações entre fluxos e com isso reduzir a taxa de bit total ocupada. No método de codificação de vídeo escalável cada fluxo produzido recebe o nome de camada. A primeira camada é chamada de **camada base** e enquanto que as demais são chamadas de **camadas de enriquecimento**. O princípio de codificação é hierárquico, sendo que cada camada de enriquecimento é responsável por acrescentar informações sobre a camada imediatamente inferior, ou seja, os vídeos são codificados (e decodificados) em seqüência da camada mais baixa para a mais alta (MARPE; WIEGAND; HERTZ, 2006).

Somente a camada base é um fluxo autocontido, que compõe o vídeo de menor resolução e taxa de exibição transmitidas do sistema. Para exibição completa da segunda camada de vídeo deve-se inicialmente decodificar quadros da camada base. Sobre estes quadros decodificados, as informações da segunda camada são acrescentadas, produzindo-se assim um vídeo com mais detalhes. A terceira camada, por sua vez, acrescenta informações sobre quadros produzidos pelo processamento da segunda camada. Ou seja, deve-se inicialmente decodificar e gerar quadros da camada base, sobre estes montar os quadros enriquecidos pela segunda camada e a partir destes pode-se compor os quadros da terceira camada. Este princípio se repete para sistematicamente conforme o número de camadas.

Por lidar nativamente com o conceito de enriquecimento de camadas, cada fluxo contém apenas as informações necessárias para aprimorar o vídeo anterior, evitando assim a transmissão de informações redundantes.

Com isso pode-se dizer que o método da escalabilidade aplicada sobre algoritmos de codificação de vídeo otimiza, de forma inteligente, a banda total ocupada para cada dispositivo, o que representa uma grande vantagem sobre o método *simulcast*. Para aplicações comerciais o consumo de banda de comunicação é um recurso muito relevante. Assim, em diversas situações, torna-se indesejável ou mesmo inviável a adoção do método *simulcast*, principalmente quando o número de fluxos de vídeo for muito grande (SULLIVAN; WIEGAND, 2007).

A Figura 3 ilustra, de forma representativa, as diferenças entre os mecanismos de codificação *simulcast* e escalável, considerando-se a banda de transmissão ocupada para três vídeos de resoluções distintas (320x240, 640x480 e 800x600 respectivamente). Para o caso dos três vídeos do exemplo, o mecanismo *simulcast* teoricamente provoca uma ocupação de 3,5Mbps contra 2Mbps do vídeo escalável.

O objetivo da figura é demonstrar de forma simplificada a diferença principal dos dois métodos. Na prática, para uma comparação quantitativa adequada, devem-se considerar diversos parâmetros práticos, tais como tipo de codificador, parâmetros de configuração e características dos vídeos. Isso porque todos estes parâmetros influenciam na qualidade e taxa de compressão dos vídeos decodificados, em relação aos algoritmos internos de cada um dos dois métodos, porém, de forma geral, as soluções escaláveis continuarão propiciando uma redução da banda ocupada.

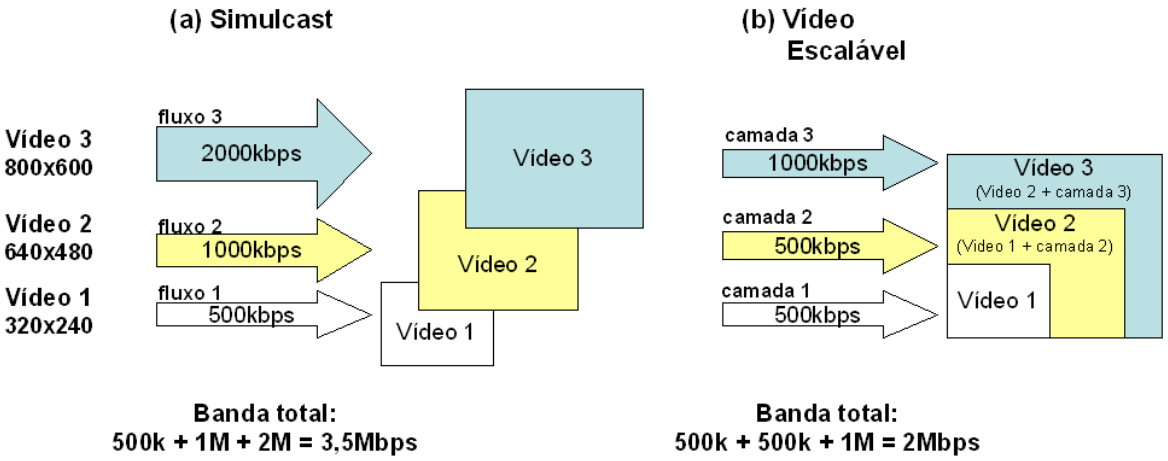


Figura 3: Comparação entre métodos de codificação (a) *simulcast* e (b) escalável.

Além da vantagem citada, o recurso da escalabilidade torna o sistema de recepção mais robusto a flutuações por problemas na rede de transmissão. No caso de algum canal de comunicação sofrer momentaneamente um erro, que provoque a perda de parte de um dos fluxos escaláveis, o vídeo pode ser decodificado até a camada mais alta intacta.

Por exemplo, se, em um canal que comporte originalmente dois fluxos escaláveis, a informação da camada de enriquecimento for corrompida, ainda assim isso não impediria a exibição da camada base. Quanto maior o número de camadas utilizado, maior pode ser adaptabilidade do sistema para flutuações na rede.

Conforme a disponibilidade tecnológica, o recurso de escalabilidade pode se beneficiar do conceito de reserva de banda ou qualidade de serviço (QoS - *Quality of Service*) buscando garantir integridade das camadas desejadas em cada ponto da rede fio (RUHIO-LOYOLA; SALA; ALI, 2008).

Assim o sistema multimídia escalável estará obtendo, em uma mesma aplicação, a geração de bandas de vídeo otimizadas sobre canais reservados. A perda de pacotes de camadas de enriquecimentos adicionais, para estes casos, pode ser considerada irrelevante ou mesmo nula, uma vez que cada canal de comunicação ocupará apenas as camadas que foram reservadas pelo projetista da rede.

Com o uso de parâmetros otimizados de QoS, pode-se ainda evitar que o uso do recurso de escalabilidade sobre canais compartilhados ocupe mais banda de comunicação que o desejado, para não comprometer outros serviços quaisquer, que, mesmo independentes, utilizam a mesma rede.

Estes são problemas até então comuns para aplicações de IPTV, onde o uso excessivo de banda para recepção de fluxos de vídeos acaba limitando significativamente o canal de comunicação disponível para outros serviços concorrentes.

Considerando as vantagens citadas, não é de surpreender que cada vez mais surjam adeptos para a tecnologia de vídeo escalável. De fato, o conceito da escalabilidade tem se mostrado tão relevante que já se apresenta como um recurso adicional previsto para a maioria dos padrões internacionais de codificação de vídeo (RIECKL, 2008).

2.2 TIPOS DE ESCALABILIDADE

O conceito de escalabilidade, quando aplicada sobre codificadores de vídeo, pode englobar diferentes estratégias para a composição de suas camadas, o que obviamente dependerá do tipo de aplicação a que se destina.

De uma forma geral, as principais estratégias adotadas para codificadores de vídeo escaláveis podem ser enquadradas em três categorias ou métodos principais, denominadas respectivamente como escalabilidade temporal, escalabilidade espacial e escalabilidade de qualidade ou SNR (*Signal Noise Ratio*).

2.2.1 Escalabilidade Temporal

O método talvez mais simples de escalabilidade é chamado de escalabilidade temporal (YANG; RATH; GUILLEMOT, 2007). Este método baseia-se no particionamento de informações do domínio tempo (quadros do vídeo a serem exibidos), a fim de compor as diferentes camadas. De forma prática, este método implica na separação das seqüências de vídeos em fluxos complementares, cada qual contendo uma distinta taxa de quadros por segundo.

Um exemplo didático da escalabilidade do tipo temporal pode ser observado na Figura 4. Neste exemplo, a seqüência de vídeo escalável é dividida em duas camadas complementares.

A primeira camada (base) contém somente os quadros ímpares, enquanto que a segunda camada (enriquecimento) contém os quadros complementares (pares). O dispositivo A processa as duas camadas geradas e assim consegue recuperar o vídeo com suas características originais. Já o dispositivo B processa apenas a camada base, e com isso obtém um vídeo com metade da taxa de exibição (número de quadros por segundos), pois somente recupera os quadros ímpares.

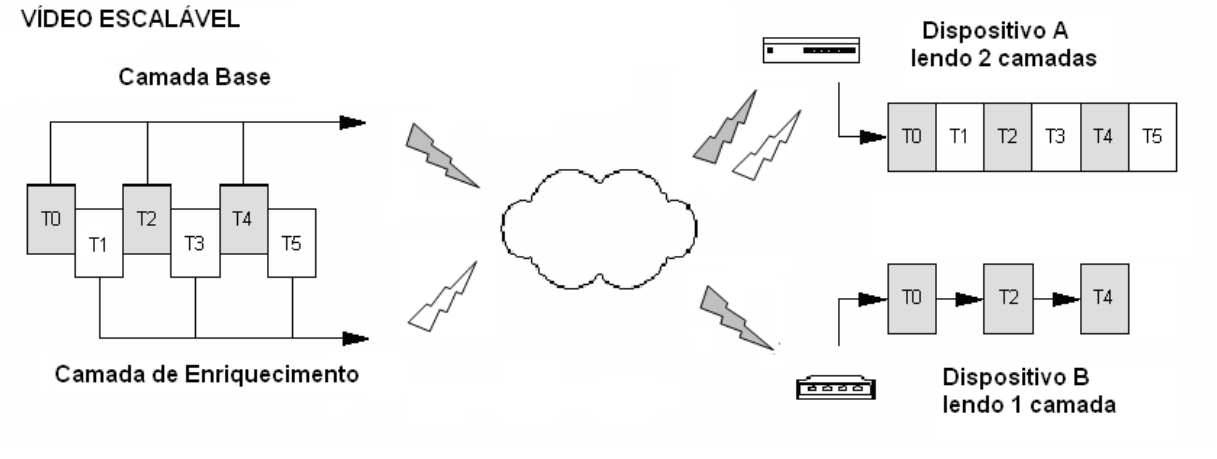


Figura 4: Exemplo de codificação escalável temporal de duas camadas.

Outra forma, um pouco mais elaborada, de se implementar o recurso de escalabilidade temporal sobre codificadores que seguem a estrutura MPEG, consiste em dividir a seqüência de vídeo em camadas compostas respectivamente por quadros do tipo I (*Intraframe*), P (*Predictive*) e B (*Bidirectionally-predictive*). A camada composta unicamente por quadros I, forma um vídeo autocontido (o vídeo que pode ser reconstruído unicamente considerando os dados presentes), representando, portanto, a camada base. A camada composta por quadros P seria complementar e utilizaria como referência os quadros anteriores (quadros I), representando assim a primeira camada de enriquecimento do sistema. A camada que contém quadros B, por sua vez, para reconstruir o vídeo, precisaria de informações dos quadros I e P, representando a camada superior de enriquecimento.

Técnicas mais sofisticadas fazem uso de transformadas multidimensionais, adotando o domínio temporal para implementação desta escalabilidade. Um exemplo é a técnica TSB (*Temporal Sub Band*) que se baseia na segmentação dos fluxos em informações obtidas pelo processamento de diferentes sub-bandas temporais independentes (PODILCHUCK;

JAYANT; FARVARDIN, 1995). A codificação ocorre considerando todo o espectro, mas os diversos fluxos de vídeo são gerados cada qual contendo parte das informações (sub-bandas). Este processo pode, entretanto, produzir um efeito colateral de borramento nos quadros, devido ao descarte de informações, que, a princípio, estariam correlacionadas e, portanto, seriam importantes para a adequada reconstrução do vídeo original.

De forma geral, a redução da taxa de quadros por segundo aumenta o tempo de reprodução de cada quadro, ocasionando *flickering* (efeito visual quando se percebe a mudança de um quadro para o outro).

2.2.2 Escalabilidade Espacial

O método de escalabilidade espacial, em contraste com o método anterior, baseia-se, no particionamento de informações dimensionais da imagem para formar as diferentes camadas. Como resultado deste método de particionamento, a camada base produz um vídeo com menor resolução espacial (dimensões horizontais e verticais de cada quadro), resolução esta que vai aumentando progressivamente, com a obtenção de informações adicionais oriundas das camadas de enriquecimento superiores (BRUNO, 2003).

Um dos mecanismos usados para implementação deste método de escalabilidade é a redução espacial por pirâmide laplaciana, que opera a partir da ponderação relativa de pixels geometricamente vizinhos. Trata-se de um método simples de ser implementado, mesmo para dispositivos de baixa complexidade, uma vez que emprega apenas somas, subtrações e deslocamentos de bits. Um exemplo do mecanismo de escalabilidade espacial pelo método de pirâmide laplaciana é apresentado na Figura 5, onde se demonstra uma aplicação para o caso de três camadas escaláveis em topologia diádica[1]. Considera-se, para ilustrar o mecanismo, o exemplo numérico de um bloco original de 4x4 pixels. Para a montagem das três camadas, inicialmente realiza-se a média aritmética de cada conjunto de 2x2 pixels, produzindo-se os valores referenciados como bloco de valores de média 1. A seguir se realiza outra média sobre estes resultados intermediários. Ao final de todas as médias obtém-se um valor de média global, também chamado de valor DC. Os valores DC representam as informações de vídeo usadas para compor a camada inferior (camada base).

[1] Denomina-se de condição diádica quando as relações entre as resoluções de duas camadas consecutivas é igual a dois

Para a montagem da segunda camada de vídeo escalável, calcula-se a diferença entre os valores de média calculados na primeira operação (Média 1) e os valores DC. Utiliza-se esta estratégia, pois, ao se calcular com resultados de diferença, estes são normalmente menores que os valores de Média 1, reduzindo assim a faixa de valores possíveis, o que aumenta a eficiência do mecanismo de compressão do codificador. O mesmo procedimento é realizado para a montagem da terceira camada de vídeo escalável (calcula-se as diferenças entre os pixels do vídeo original e os valores de Média 1). Conforme já comentado, devido à relação diádica deste método, o vídeo reconstruído por cada camada possui metade da resolução da camada imediatamente superior, o que pode ser observado à direita na figura (vídeos reconstruídos).

No exemplo apresentado não existem erros envolvidos nos processos de compressão e descompressão, porém na prática ocorrem erros de representação, devidos aos truncamentos dos resultados das médias quando se trabalha com aritméticas de ponto fixo, que podem afetar a qualidade do vídeo reconstruído.

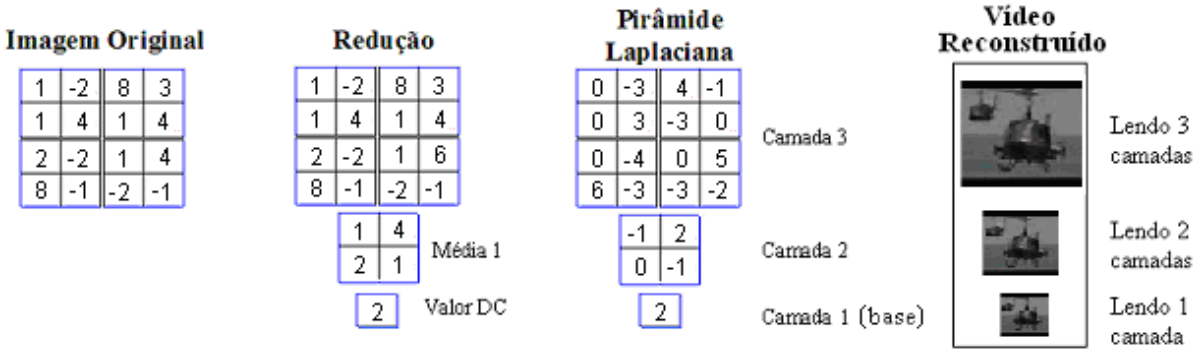


Figura 5: Escalabilidade por redução espacial baseada em pirâmide laplaciana

Outro algoritmo utilizado para efetuar a escalabilidade espacial se baseia no uso de transformadas de Wavelet (MARPE; WIEGAND; HERTZ, 2006).

Os dados resultantes de uma decomposição Wavelet podem ser, de uma forma simplificada, interpretados como coeficientes de detalhe (Dn) e aproximação (An). A aplicação reincidente da transformada Wavelet sobre os coeficientes de aproximação vai gerar novos coeficientes de Dn e An, porém com metade da resolução espacial (exemplo apresentado na Figura 6).

Neste conceito, a camada base é composta pelo primeiro conjunto de coeficientes de detalhe, enquanto que as subsequentes camadas de refinamento vão sendo formadas a partir

dos resultados de aproximação gerados respectivamente por cada estágio de transformação Wavelet. Como se pode observar na figura a cada processo de transformada a resolução dos vídeos resultantes mantém inerentemente uma relação diádica (HUANG; PENG; CHIANG, 2007).

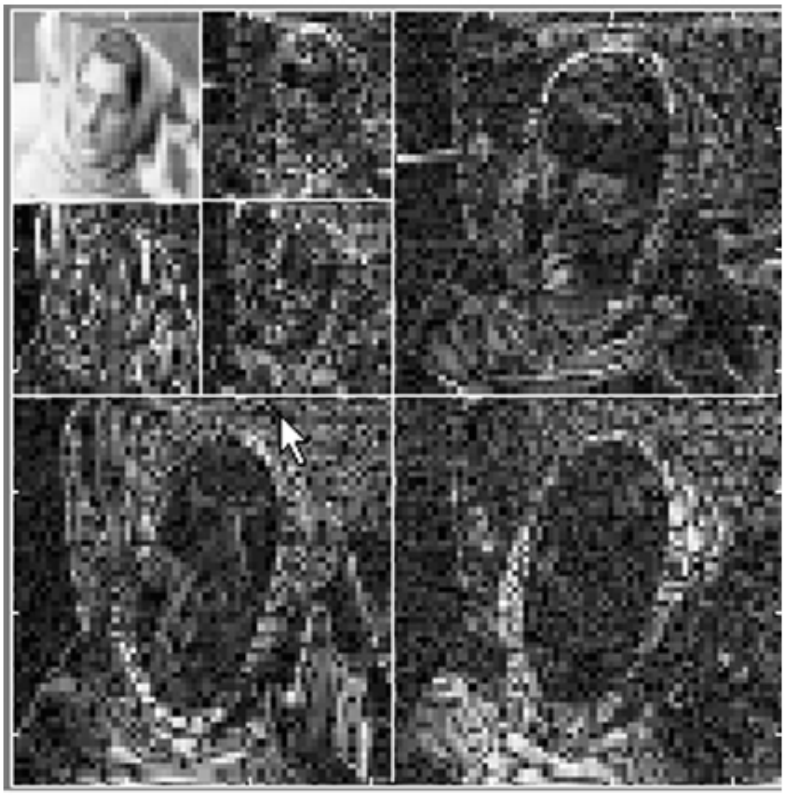


Figura 6: Uso de transformadas Wavelet para obtenção de escalabilidade espacial

2.2.3 Escalabilidade de Qualidade ou SNR

A escalabilidade de qualidade, ou também chamada escalabilidade SNR, não afeta características básicas do vídeo tais como a taxa de exibição (número de quadros por segundo) ou resolução espacial do mesmo. Ao invés disso, este método se baseia na relação de qualidade medida entre o vídeo original e o que estiver sendo reconstruído após o processamento das diversas camadas. De forma geral, o vídeo obtido a partir da camada base possui a qualidade mínima desejada para sua reprodução, conforme configuração própria definida para o processo de codificação. As diferentes camadas de enriquecimento que forem produzidas serão usadas para adicionar informações, a fim de melhorar a qualidade visual da imagem reconstruída.

Uma técnica bastante simples para a implementação da escalabilidade SNR se baseia no mecanismo de seleção de planos de bit (técnica de *bitplanes*). Neste mecanismo as camadas escaláveis são construídas de forma que cada uma delas contém partes complementares dos bits de representação da imagem. Os bits mais significativos das informações de vídeo compõem a camada base, enquanto que os demais vão sendo transmitidos como camadas de enriquecimento.

Um exemplo deste mecanismo é apresentado na Figura 7, onde são montadas quatro camadas escaláveis. No exemplo, os diversos valores codificados são representados como números binários de quatro bits, dispostos em forma matricial. A seqüência de todos os bits mais significativos (*Most Significant Bit* – MSB) constitui a informação a ser incluída na camada 1, ou seja a camada base. Os próximos bits de representação (segundo bit menos significativo), logo na seqüência, compõem a segunda camada escalável, os quais sendo concatenados com os dados da camada anterior aumentam a qualidade das informações recuperadas.

Este processo de acréscimo de bits de representação deve se repetir até a última camada, produzindo, como apresentado no exemplo em questão, um sistema escalável de quatro camadas. Conforme o caso pode-se escolher utilizar dois ou mais bits de representação em cada camada, o que vai depender de fatores como taxa desejada de bits por segundo, número de camadas e da mínima qualidade escolhida, entre outros fatores (BRUNO, 2003).

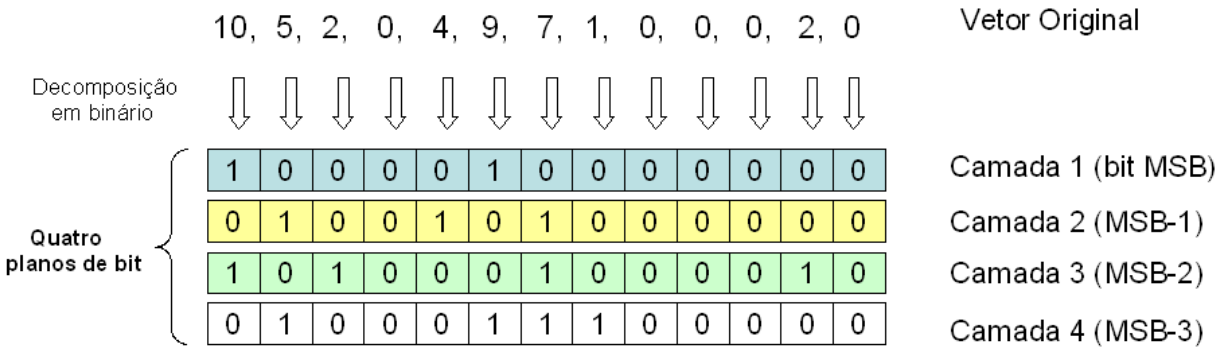


Figura 7: Exemplo do uso de planos de bit para obtenção de escalabilidade SNR

Após a divisão da informação dos diferentes planos de bits os dados resultantes são comprimidos por técnicas de entropia, que utilizam códigos de representação próprios a fim de reduzir o número de bits ocupados.

De certa forma, esta estratégia de dividir os valores em diferentes resoluções de bit pode ser interpretada com um mecanismo de quantização variável. Na realidade, métodos recentes de escalabilidade SNR valem-se exatamente disso, ou seja, atuam diretamente na precisão dos módulos internos de quantização dos codificadores de vídeo, provocando assim uma variação controlada na qualidade dos quadros codificados (RIECKL, 2008). Os erros causados pelo aumento da intensidade dos quantizadores vão sendo progressivamente corrigidos pelas camadas escaláveis superiores, que, a cada refinamento, aumentam a precisão dos dados recuperados.

Outra estratégia, interpretada como escalabilidade SNR, é obtida pela segmentação ou particionamento de dados no domínio freqüência. A implementação deste método se aplica sobre os dados oriundos de um processo de transformada no codificador, quando os dados da imagem estariam sendo representados no domínio freqüência. O mecanismo de segmentação de camadas ocorre escolhendo-se um determinado ponto no bloco de coeficientes resultantes das transformadas, a partir do qual os demais coeficientes serão descartados para a respectiva camada.

Para ilustrar este procedimento, pode-se considerar o exemplo da Figura 8. O bloco ilustrado representa uma matriz de 8x8 coeficientes resultantes de uma transformação. A transformação bidimensional tem por característica agrupar as informações de baixa freqüência na posição superior esquerda e as de alta freqüência na posição inferior direita. De fato a leitura dos dados de uma matriz resultante de transformada, caso deseje-se acompanhar as informações em ordem crescente no domínio das freqüências, deve ser feita em uma forma de zig-zag, como indicado na figura.

O codificador escalável escolhe um ponto nesta seqüência de coeficientes, até onde devem ser incluídos na camada base. Este ponto é chamado de ponto de particionamento. Os valores encontrados após o ponto de particionamento devem ser transmitidos em outra(s) camada(s) (DARONCO, 2008).

Pode-se perceber que, como a imagem reconstruída a partir da camada base perdeu informações de freqüências mais elevadas, esta deverá apresentar um efeito de perda de detalhes ou nitidez, principalmente em pontos da imagem onde deveriam ocorrer variações bruscas de cor. As camadas de enriquecimento trazem os coeficientes descartados pela camada base melhorando a qualidade do vídeo reconstruído.

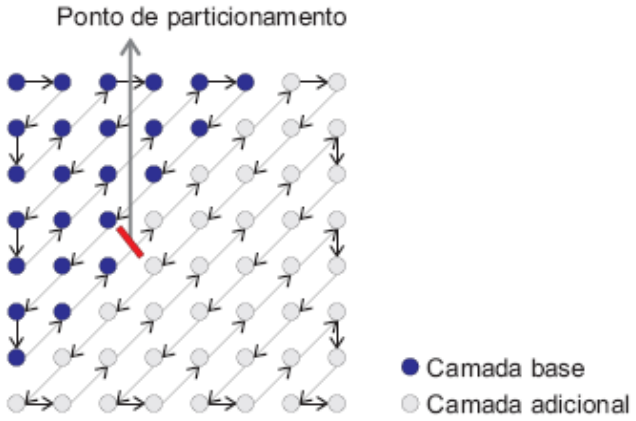


Figura 8: Uso da técnica de particionamento de dados de freqüência

(DARONCO, 2008)

Essa técnica de particionamento é adotada como mecanismo de visualização gradual do formato de representação de imagens denominado como JPEG progressivo. Neste mecanismo, o software de exibição primeiramente tem acesso aos coeficientes de baixa freqüência, que são usados para gerar uma visualização prévia da imagem. A seguir o software decodificador vai processando os demais coeficientes, e por conseqüência aumentando progressivamente a nitidez da imagem até atingir sua qualidade final.

2.2.4 Escalabilidade Híbrida

Apesar da distinção clara entre os tipos de escalabilidade (temporal, espacial e SNR), na prática, muitas vezes se observam demandas divergentes que dificultam a seleção de um destes métodos em detrimento dos demais. Na verdade, cada um dos tipos de escalabilidade anteriormente citados se adequa melhor para uma dada aplicação.

A escalabilidade espacial, por exemplo, é especialmente vantajosa para atender a redes contendo dispositivos com diferentes capacidades de exibição de vídeo, seja por características de limitação no visor ou poder de processamento.

A escalabilidade temporal, por sua vez, é útil para aplicações onde a nitidez é importante, mas não tanto a taxa de atualização de quadros por segundo.

Já a escalabilidade SNR é indicada para aplicações sujeitas a canais de comunicação pouco confiáveis, mas que visem funcionamento com demandas de tempo-real.

De fato o recurso de escalabilidade aplicado sobre algoritmos de codificação de vídeo é um problema multi-dimensional. Padrões escaláveis mais flexíveis suportam a combinação simultânea de diferentes tipos de escalabilidade, visando melhor se adequar a heterogeneidade de redes e dispositivos. Quando dois ou mais métodos de escalabilidade são utilizados simultaneamente, diz-se que o sistema adota o conceito de escalabilidade híbrida (BRUNO, 2003). A figura a seguir traz um exemplo de escalabilidade híbrida, onde se adota pelo menos três níveis espaciais (QCIF, CIF e 4CIF), três níveis temporais (7,5 fps, 15 fps e 30 fps) e dois níveis SNR (Q0 e Q1), produzindo um sistema escalável de 18 camadas. A camada base contém a menor quantidade de informação em todos os níveis (T0, S0 e Q0).

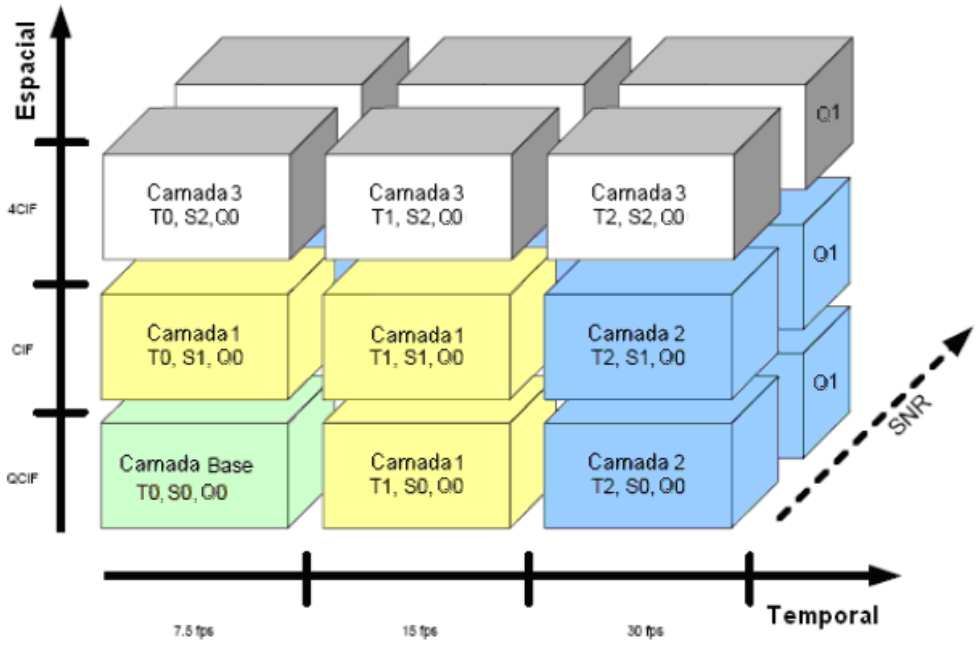


Figura 9: Modelo multidimensional de escalabilidade híbrida (18 camadas)

2.3 CODIFICADORES DE VÍDEO ESCALÁVEIS

Conforme já comentado, diversos algoritmos de codificação de vídeo têm sido definidos nas últimas décadas visando transmissão ou mesmo armazenagem. De forma simplificada, pode-se dizer que o aprimoramento continuado destes diferentes padrões culminou com a especificação H.264/SVC (SULLIVAN; WIEGAND, 2005).

2.3.1 Evolução dos Codificadores de Vídeo Não Escaláveis

Para auxiliar a entender a evolução das normas de codificação de vídeo, abaixo se apresenta, em uma linha temporal, as épocas de lançamento dos diferentes padrões definidos pelas entidades MPEG e ITU (*International Telecommunication Union*).

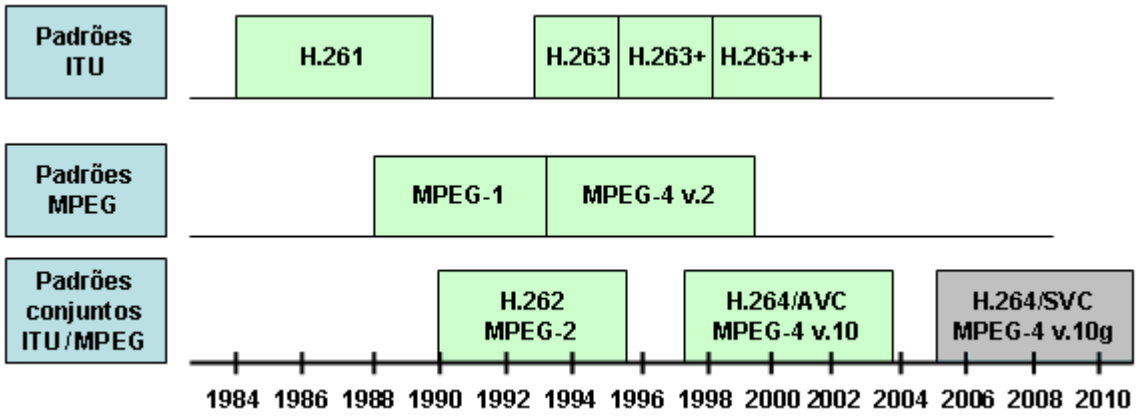


Figura 10: Evolução das normas de codificação de vídeo

A norma H.261 de meados dos anos 80 foi a primeira iniciativa da organização ITU de gerar um padrão para aplicações de teleconferência. Alguns anos depois, o grupo MPEG lançou o padrão MPEG-1 visando principalmente aplicações de armazenamento de vídeo. Posteriormente as duas organizações lançaram em conjunto a especificação H.262/MPEG-2 (H.262 pela entidade ITU e MPEG-2 pela organização MPEG), que veio a se tornar o padrão mais difundido para aplicações de vídeo digital, sendo usado por exemplo como padrão de armazenamento em DVD e de codificação para TV digital (KEITH, 2004).

Em 1993, a organização ITU chegou a especificar a norma H.263, como substituta do padrão H.262/MPEG-2, visando trabalhar com quadros de maior definição, porém com poucos ganhos em relação ao H.262/MPEG-2. Estes pequenos ganhos registrados na relação custo x benefício fez com que o grupo MPEG nem lançasse a especificação MPEG-3 (NGUYEN; TAN, 2006).

No final dos anos 90, o grupo MPEG lançou a especificação MPEG-4 buscando viabilizar a transmissão de vídeos de alta definição com maior taxa de compressão. Em paralelo a este desenvolvimento, a organização ITU vinha trabalhando em um projeto de codificação ainda mais eficiente. As duas organizações então uniram esforços para lançar outro padrão conjunto, o qual constitui hoje a mais moderna especificação de codificação de

vídeo não escalável destas entidades. A organização ITU intitulou esta especificação como H.264, enquanto que o grupo MPEG apresentou a mesma como uma extensão da especificação, chamada de MPEG-4 parte 10 ou MPEG-4 AVC (*Advanced Video Coding*).

Uma das principais melhorias implementadas pelo H.264 está no módulo de transformada DCT (*Discrete Cosine Transform*) do codificador, que passou a adotar um algoritmo que utiliza apenas números inteiros sem multiplicações. Este algoritmo aplicado sobre blocos de 4x4 é bastante rápido e totalmente reversível, sendo, portanto ideal para implementações em hardware (RICHARDSON, 2003).

Outro avanço da norma H.264 está na inclusão de um mecanismo de predição intra. Assim é possível se explorar, dentro de um mesmo quadro, a redundância espacial ao se identificar a similaridade entre blocos dentro da mesma imagem. Utilizando-se blocos de 4x4 pixels pode-se fazer a predição intra em até nove modos direcionais, conforme indicado na Figura 11. Blocos 16x16 também podem ser usados, porém, neste caso, apenas quatro direções são possíveis.

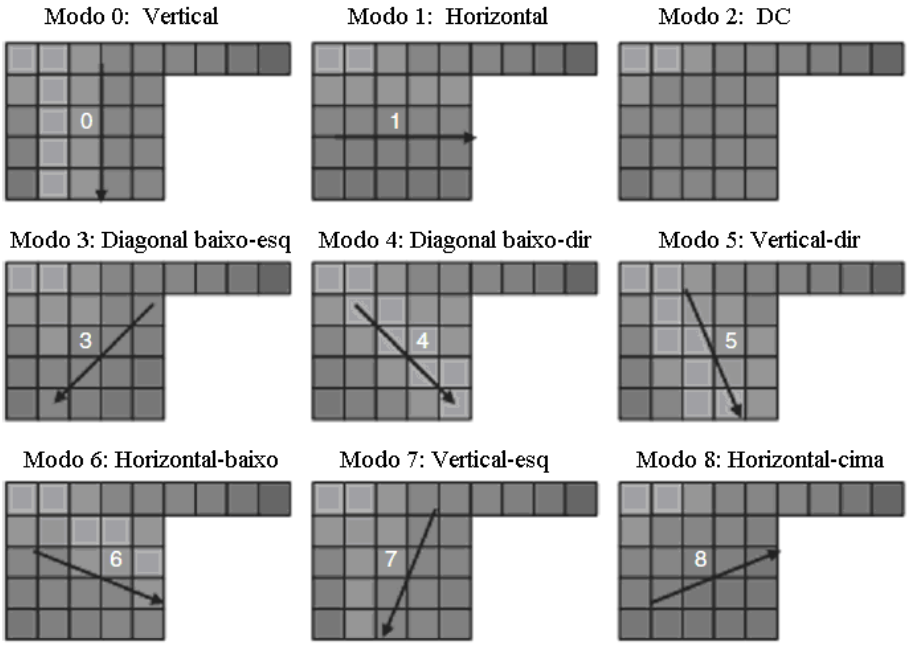


Figura 11: Modos de predição possíveis para blocos intra 4x4

Adaptado de (CHEN et al. 2009)

Já na predição entre quadros a especificação H.264 prevê a utilização de múltiplos quadros de referência, o que torna este padrão mais flexível que as normas anteriores que somente suportam até duas referências.

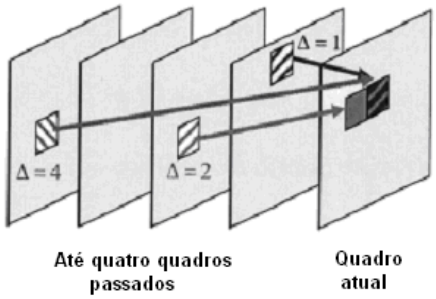


Figura 12: Predição com suporte a múltiplos quadros de referência
(KUO; CHAN, 2006)

Ainda no campo da predição de movimento, a norma H.264 inovou prevendo o particionamento dos blocos de pixels em diversas configurações tais como 4x4, 4x8, 8x4, 8x8, 8x 16, 16x8 e 16x16 (KUO; CHAN, 2006). Este recurso permite uma codificação mais flexível e um procedimento de predição de movimento com menor incidência de erro (resíduo). A Figura 13 compara resultados do particionamento com o método tradicional.

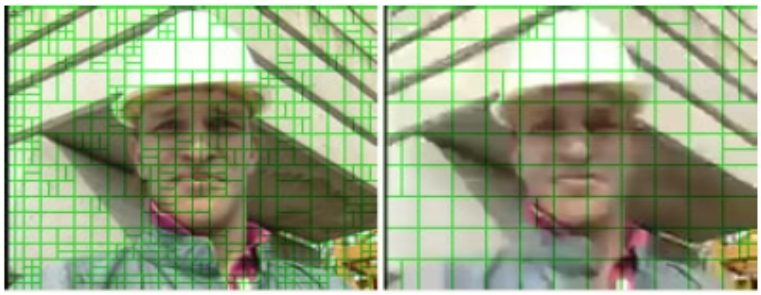


Figura 13: Divisão da imagem para estimativa de movimento com e sem particionamento

A especificação do codificador H.264 previu ainda um algoritmo adicional de filtro anti-blocagem. Este filtro aplicado sobre a imagem melhora consideravelmente a qualidade do vídeo exibido. O processo de filtragem adotado provê um alto grau de adaptabilidade desde falhas de continuidade entre bordas de blocos de pixels vizinhos. Como resultado a qualidade subjetiva é significativamente melhorada reduzindo a taxa de bit ocupada. Considerando-se a mesma qualidade de vídeo percebe-se em média uma redução de 5 a 10% da taxa a favor da versão com filtragem habilitada (RICHARDSON, 2003).

No campo da entropia, o codificador H.264 pode fazer uso de um mecanismo de tamanho de código variável adaptativo ao contexto do fluxo (*Context Adaptive Variable*

Lenght Coding - CAVLC). O mecanismo CAVLC comuta entre diferentes tabelas VLC procurando o resultado que melhor se adapta a determinada característica de vídeo (contexto).

Além do CAVLC, o H.264 prevê outro mecanismo, CABAC (*Context Adaptive Binary Arithmetic Coding*), que ao contrário das versões anteriores não trabalha diretamente sobre os elementos sintáticos, mas sim sobre o fluxo de bits de saída (PASTUSZAK, 2006).

Trata-se de uma codificação aritmética binária que busca levantar o modelo probabilístico mais adequado para representar o próximo bit. Neste modelo, a tabela de contexto é dinamicamente atualizada conforme o fluxo é analisado. Comparando-se com CAVLC este mecanismo mais complexo pode gerar um ganho de 5 a 15%, como pode ser visto na Tabela 1.

Tabela 1 - Ganhos obtidos com novos algoritmos do H.264

Algoritmo	Ganho obtido
Predição espacial para MB tipo Intra	Reduz 6-9% bits
Predição temporal com particionamento	Reduz aproximadamente 50% bits
Transformadas de inteiros	PSNR perde menos de 0.02dB
CAVLC	Reduz 5-8% bits
CABAC	Reduz 5-15% bits sobre CAVLC
Filtro Deblocking	Reduz 5-10% bits

Fonte: Richardson (2003).

Uma análise comparativa entre estes modelos de codificadores de vídeo, demonstrada na Figura 14, confirma que a especificação H.264 (H.264/MPEG-4 AVC) é a que apresenta, mantendo uma mesma qualidade de vídeo, o melhor desempenho de compressão em comparação aos demais padrões citados: MPEG-2, H-263 e MPEG-4.

Em destaque, conforme apresentado no gráfico, pode-se observar que, para um mesmo valor de PSNR, *Peak Signal Noise Ration,* (no caso PSNR = 36 dB), o codificador H.264 ocupa cerca de 40% da taxa de bits exigida pelo difundido padrão MPEG-2, o que representa, em termos de eficiência de compressão, em um ganho de aproximadamente 60% (RICHARSON, 2003).

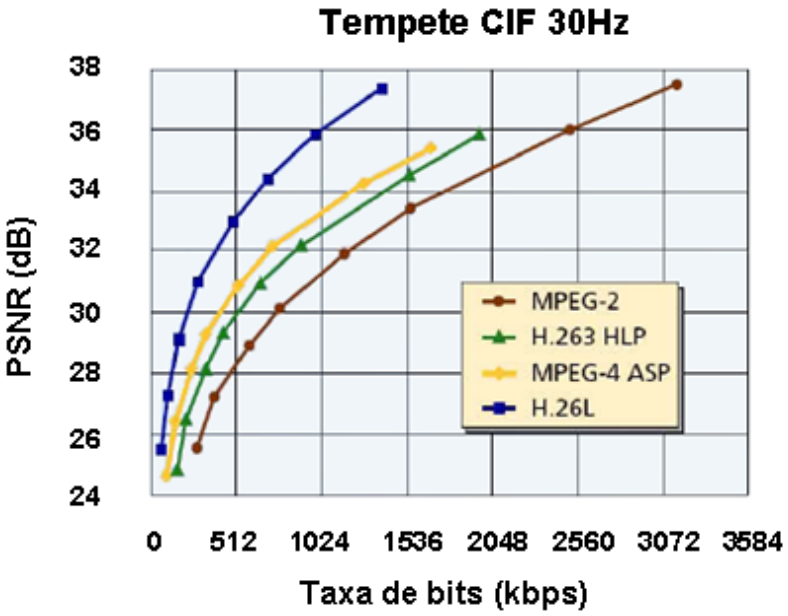


Figura 14: Gráfico comparativo da eficiência de codificadores de vídeo.

2.3.2 Evolução dos Codificadores de Vídeo Escaláveis

Nos últimos anos, as organizações MPEG e ITU especificaram diferentes perfis ou extensões escaláveis para as especificações anteriormente citadas. Uma análise evolutiva destas permite de forma simplificada dividir as especificações em cinco gerações, a seguir descritas.

2.3.2.1 Primeira Geração: Especificações H.261 e MPEG-1

A especificação de codificação de vídeo denominada H.261 (1984) foi a primeira iniciativa da organização ITU-T visando gerar um padrão para aplicações de teleconferência. Pouco tempo depois (1989) o grupo MPEG lançava o padrão MPEG-1, fortemente baseado no padrão H.261, visando aplicações de teleconferência e armazenamento de vídeo em disco. Os padrões H.261 e MPEG-1, por se tratarem de especificações antigas de codificação de vídeo, não tinham foco em flexibilidade, e assim sendo não previram o recurso de escalabilidade. Assim sendo, para permitir adaptação ao meio dos padrões H.261 ou MPEG-1 a única solução possível é fazer uso do método *simulcast*.

2.3.2.2 Segunda Geração: Especificação H.262/MPEG-2

Em 1992, o padrão H.262/MPEG-2, foi criado como uma especificação conjunta entre as organizações MPEG e ITU (KEITH, 2004). Este padrão foi desenvolvido em consenso com o mercado de difusão global (*broadcast*), suportando compatibilidade retroativa, ou seja, sua versão básica é compatível com MPEG-1.

Como vantagem adicional, o padrão H.262/MPEG-2 foi o primeiro a prever a implementação de codificação em camadas, fazendo referência a uma camada base. Assim sendo, pode-se dizer que H.262/MPEG-2 foi o primeiro padrão de codificação de vídeo, de propósito geral, que incluiu em sua especificação o recurso de escalabilidade. Foram previstos nesta norma os principais tipos de escalabilidade (temporal, espacial, SNR e híbrida).

Sua especificação traz, porém a restrição de suportar somente duas camadas para cada tipo de escalabilidade. Como conceito de escalabilidade híbrida, a norma permite que se adote, em uma mesma solução, dois tipos quaisquer de escalabilidade (HUSEMANN, 2006a). No caso mais completo, são possíveis três camadas escaláveis (uma camada base e uma camada de enriquecimento para cada tipo de escalabilidade escolhida).

A escalabilidade temporal prevista na norma MPEG-2 utiliza quadros B como camadas de enriquecimento (CHIEN et al., 2005). Trata-se de uma forma simples de implementação de escalabilidade temporal. A escolha deste mecanismo se deve, principalmente, ao fato destes quadros não poderem ser utilizados como referência para outros quadros na norma MPEG-2. Assim sendo, podem ser perdidos sem comprometer a decodificação do vídeo como um todo (quadros da camada base devem ser autocontidos).

A escalabilidade espacial implementada na norma MPEG-2 é um pouco mais complexa, suportando duas camadas: camada base, de baixa resolução e camada de enriquecimento, contendo informações complementares para aumento de resolução.

A relação entre as resoluções da camada base e da imagem final não precisa ser múltiplo de dois (caso diádico) ou mesmo inteira. De fato, o processo de escalabilidade espacial da norma MPEG-2 é bem flexível na seleção do trecho de vídeo a ser usado para compor a camada base, suportando diferentes configurações para definição da área de vídeo da camada base: dimensões de largura e altura da imagem, deslocamentos vertical e horizontal do ponto de corte (*cropping*).

A norma permite inclusive operação com imagens que possuam diferentes resoluções de aspecto[2] de vídeo (como, por exemplo, os aspectos 4:3 ou 16:9).

Uma ilustração das possíveis configurações desta seleção é apresentada na Figura 15, onde se mostra um exemplo em que o vídeo obtido pela camada base tem resolução SD (720x480 pixels) e o vídeo reconstruído após processar a camada de enriquecimento tem resolução HD (1920x1080).

Pode-se perceber que devido às resoluções de aspectos serem distintas (um vídeo SD tem aspecto 4:3 enquanto que um vídeo HD tem aspecto 16:9), mesmo expandindo-se o quadro da camada base, muitas informações de borda devem ser acrescentadas para compor-se o seu formato de maior resolução.

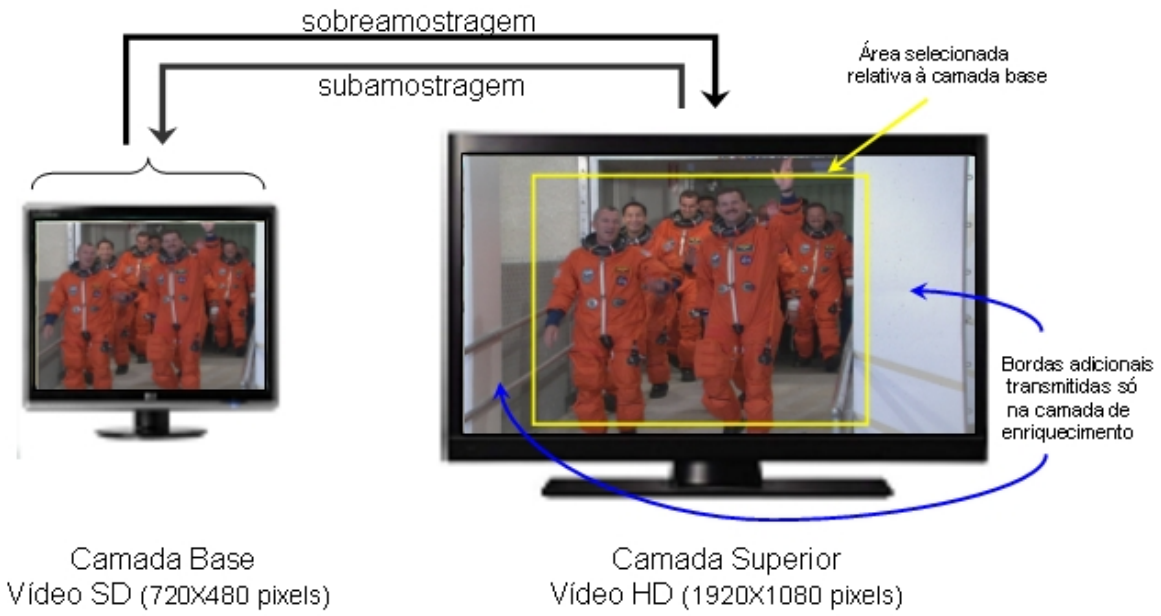


Figura 15: Exemplo de relação de camada base SD sobre vídeo HD

No processo de codificação com escalabilidade espacial a primeira etapa é selecionar, a partir da imagem original de alta resolução, a região a ser adotada para gerar a camada base. A resolução espacial da área selecionada não precisa necessariamente ser a mesma de exibição. Para tanto é prevista na norma a aplicação de um filtro adicional de subamostragem que converte as resoluções.

[2] A resolução de aspecto de um dado vídeo representa a relação entre suas dimensões horizontal e vertical. Normalmente é dada por dois valores inteiros (divisor e dividendo), cuja relação é a mesma presente no vídeo.

Os critérios de intensidade do filtro são definidos segundo parâmetros como fatores de multiplicação horizontal e vertical, bem como deslocamentos horizontais e verticais do ponto inicial da área selecionada.

A imagem de baixa resolução (obtida após subamostragem) é então codificada formando a camada base. Os dados codificados passam por um processo de reconstrução, a fim de obter as mesmas informações que um decodificador teria.

A imagem reconstruída é então reescalonada (processo de sobreamostragem), para recuperar o tamanho e posições originais. Esta imagem recuperada é também chamada de imagem de predição (no exemplo da figura anterior seria a área destacada).

Todo este procedimento é regido pela norma H.262, obedecendo a um algoritmo específico.

No método de escalabilidade SNR, o fator usado para composição das duas camadas é o parâmetro de quantização do codificador, o qual atua diretamente na qualidade do vídeo codificado. A camada base é a que tem maior valor de intensidade do quantizador, produzindo uma imagem de menor qualidade.

A codificação da camada de enriquecimento utiliza como imagem de predição a imagem de menor qualidade gerada após reconstrução da camada base.

A seguir são calculados os resíduos de pixels da imagem de predição em relação à imagem original.

Os resíduos são codificados com um quantizador menos severo que o usado pela camada base, gerando assim a camada de enriquecimento, que será usada para melhorar a qualidade do vídeo final.

Um exemplo do princípio de funcionamento deste mecanismo para duas camadas é apresentado na Figura 16.

Figura 16: Escalabilidade SNR na norma MPEG-2

2.3.2.3 Terceira Geração: Especificação H.263

Em 1993, a organização ITU lançou a especificação a norma H.263, visando principalmente trabalhar com quadros de maior definição. O objetivo é que esta nova especificação substituísse a norma H.262/MPEG-2 vigente, porém este objetivo não se processou, devido à grande base instalada do MPEG-2 e ao pequeno ganho de desempenho obtido. Em contraste com sua antecessora, a norma H.263 não incluiu o recurso de escalabilidade em sua especificação inicial.

Entretanto, posteriormente ao seu lançamento foram definidas extensões da norma H.263 que previam suporte às escalabilidades do tipo temporal, espacial e SNR, chamado de H.263+ (NGUYEN; TAN, 2006).

2.3.2.4 Quarta Geração: Especificação MPEG-4

No final dos anos 90, o grupo MPEG lançou a especificação MPEG-4, uma norma bem mais flexível que as anteriores buscando trabalhar com transmissão de vídeos de alta definição com maior taxa de compressão, mas também visando o mercado incipiente de dispositivos móveis (HE, 2004).

A norma MPEG-4, em seu perfil mais simples, é equivalente ao H.263, ou seja, não suporta o recurso de escalabilidade. Perfis mais complexos da norma, entretanto, incluem o recurso de vídeo escalável, trazendo inclusive algumas vantagens sobre as implementações anteriores. Como vantagens pode-se destacar inicialmente que o padrão MPEG-4 adota um modelo inovador de implementação, que possibilita a obtenção simultânea de informações em dois domínios distintos, trazendo assim, nativamente, o recurso de escalabilidade híbrida (CHEN; PENG, 2002).

Outra característica única da especificação escalável da norma MPEG-4 é que a mesma suporta a implementação prática de escalabilidade otimizada no nível de objetos de vídeo (VOL – *Video Object Layer*), o que representa um modelo mais flexível do que a divisão tradicional de dados em quadros de vídeo.

Além disso, esta nova norma previu um conceito inovador de granularidade fina, que foi projetado visando uma implementação muito mais sensível às adaptações dinâmicas de

rede. Este formato é especialmente sugerido para suportar a interligação do codificador com algoritmos atuais de controle de congestionamento em redes IP.

O novo mecanismo, também chamado de FGS (*Fine Grain Scalability*), pode ser aplicado sobre a codificação escalável de qualidade (SNR), temporal, ou híbrida, considerando-se que se utilize simultaneamente camadas temporais e SNR (RADHA; SCHAAR; CHEN, 2001).

A estrutura de FGS consiste basicamente na construção de duas camadas: uma camada base, com taxa de bit constante, e uma camada de enriquecimento, com taxa de bit variável, limitada a um valor máximo previamente configurado. A construção da camada base é similar às demais implementações.

Para a montagem da camada de enriquecimento, primeiramente deve-se considerar a taxa de bit máxima programada. Posteriormente o codificador tem flexibilidade para definir, segundo dados do meio físico, a quantidade destes dados que vai de fato ser enviada. Assim a determinação da banda ocupada ocorre no estágio de montagem do fluxo de bit e não no mecanismo de quantização.

Com esta estratégia elimina-se a necessidade da execução de mecanismos complexos e invasivos de controle da taxa de bit, permitindo a manipulação de sessões de atendimento *unicast* adaptadas dinamicamente às variações da largura de banda em tempo real.

Quando o modelo FGS foi introduzido pela primeira vez no MPEG-4, foram analisadas duas propostas para codificação da camada de enriquecimento: método de compressão por Wavelet ou mecanismo baseado em planos de bit sobre DCT. Os resultados de ensaios considerando diferentes variações da codificação para ambas as técnicas (codificação por planos de bit e compressão baseada em Wavelet) produziram resultados de desempenho muito similares (SULLIVAN; WIEGAND, 2005). Pesou contra a abordagem por transformada Wavelet o fato deste método exigir grande quantidade de memória no receptor, o que é crítico para os dispositivos de baixa complexidade.

Além disso, devido ao fato da camada base FGS já ser codificada usando um mecanismo compatível com MPEG-4 DCT, a escolha do método DCT para compressão da camada de enriquecimento se mostrou como a opção mais vantajosa. Sendo assim a escalabilidade fina MPEG-4 adota um mecanismo de planos de bit sobre coeficientes DCT para a construção de múltiplas camadas.

Usando a transformada DCT nas duas camadas (base e enriquecimento) o cálculo de resíduo entre camadas fica simplificado.

Os vetores de saída são montados a partir dos blocos resultantes da DCT, iniciando do plano de bit mais significativo e encerrando com o plano de bit menos significativo (Figura 17).

Para cada plano de bit os macroblocos são compostos por quatro blocos de luminância e dois blocos de crominância.

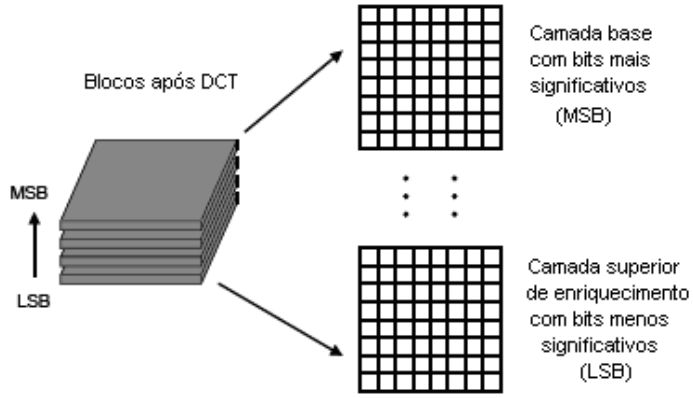


Figura 17: Estratégia de leitura de macroblocos usando planos de bit

O algoritmo de entropia VLC (*Variable Lenght Coding*) utiliza códigos de comprimento variável para compressão sem perda. Um código especial “todos os blocos são zero” é usado quando todos os seis blocos de planos de bit não têm qualquer bit com valor um.

Todas as melhorias do FGS já haviam sido adotadas pelo padrão de vídeo MPEG-4. Embora promissor, entretanto este mecanismo de granularidade fina não apresenta uma boa eficiência de compressão.

Algoritmos mais modernos confiam na compensação de movimento para melhorar a eficiência da codificação ao custo de sacrificar a flexibilidade e o tratamento de perdas de pacote.

Soluções que evitam a compensação de movimento são mais rápidas, porém, conseqüentemente, reduzem muito a eficiência da codificação (RADHA; SCHAAR; CHEN, 2001).

2.3.2.5 Quinta Geração: Especificação H.264

Assim como aconteceu com a especificação H.263 em sua primeira versão, a norma H.264 também não previa o uso da escalabilidade. Após sua especificação inicial, porém, pesquisadores do grupo de trabalho denominado JVT (*Joint Vídeo Team*) assumiram a tarefa de propor a versão escalável do codificador H.264.

O processo de definição desta nova especificação foi longo e exaustivo, pois se desejava produzir uma especificação inovadora que trouxesse diversas vantagens sobre as alternativas anteriores (LI; LIU, 2003).

O grupo JVT iniciou, em outubro de 2003, a chamada de propostas para tecnologias eficientes de codificação escalável, visando desenvolver um novo padrão chamado SVC (*Scalable Video Coding*).

No ano de 2004, foram apresentadas 14 propostas de algoritmos escaláveis de diferentes instituições de pesquisa e universidades.

Doze das 14 propostas submetidas utilizavam técnicas baseadas em transformadas Wavelet, enquanto que as duas remanescentes eram fortemente baseadas no modelo H.264/AVC.

Em janeiro de 2005, o projeto SVC começou a ser escrito como um adendo da norma H.264/AVC.

Além das soluções anteriormente utilizadas por codificadores escaláveis, o padrão passou a incluir uma estrutura de decomposição Wavelet em direção temporal. Posteriormente, esta estrutura deixou de ser obrigatória na especificação SVC, visando reduzir a complexidade do codificador e decodificador.

Em março de 2007, por fim, concluiu-se esta especificação, que passou a ser referenciada como H.264 SVC para contrastar com a versão simples (não escalável) da norma H.264 AVC.

A especificação SVC foi incluída como parte integrante da norma H.264 como regulamentação anexa, no mês de novembro do mesmo ano (HUANG; PENG; CHIANG, 2007).

Como resultado, pode-se dizer que esta especificação representa atualmente a mais moderna e completa proposta de escalabilidade para codificadores de vídeo.

3 PADRÃO ESCALÁVEL SVC DA NORMA H.264

Apesar da clara importância do recurso de escalabilidade em codificadores de vídeo, as especificações escaláveis dos padrões H.262/MPEG-2, H.263 e MPEG-4 não foram bem aceitas pelo mercado (WIEN; SCHWARZ; OELBAUM, 2007). É fato que para possibilitar uma utilização prática qualquer solução de codificador deve atender a um compromisso entre capacidade de codificação e desempenho. Um dos problemas antigos dos codificadores escaláveis reside em sua reduzida eficiência de codificação, ou seja, o vídeo reconstruído por uma solução escalável acaba registrando, para uma mesma taxa de bits, perdas significativas de qualidade quando comparado com o vídeo resultante de sua versão não escalável. Versões mais recentes de codificadores escaláveis procuram incluir algoritmos que aumentem a sua eficiência de codificação, por exemplo, aumentando sua flexibilidade nos módulos de predição entre camadas ou mesmo incluindo filtros de pré e pós-processamento. Estas soluções geralmente melhoram a qualidade do vídeo produzido, mas acabam aumentando a complexidade do codificador, bem como a latência de codificação.

Foi neste contexto que pesquisadores das entidades MPEG e ITU buscaram unir seus esforços para o desenvolvimento de um novo projeto de codificador escalável, compatível com o padrão H.264/MPEG-4 AVC. Ao contrário das demais propostas escaláveis este novo projeto (H.264/SVC) foi trabalhado com o apoio de profissionais da indústria visando, ao mesmo tempo, reunir soluções que garantissem grande eficiência de compressão, reduzida complexidade no codificador e baixo tempo de latência. Após quatro anos de pesquisa, em janeiro de 2007, a proposta foi publicada em sua versão final como um adendo do padrão H.264/AVC (MARPE; WIEGAND; HERTZ, 2006).

3.1 IMPLEMENTAÇÃO SVC DOS DIFERENTES TIPOS DE ESCALABILIDADE

A especificação H.264/SVC atende a várias aplicações, tais como conteúdo multiresolução, sendo previstos todos os tipos de escalabilidade (temporal, espacial, SNR e híbrida) com um grande número de camadas produzidas (HUANG; PENG; CHIANG, 2007). A fim de facilitar o entendimento destes algoritmos conforme especificados no padrão SVC, a seguir serão apresentados separadamente os mecanismos utilizados para implementação de cada um dos tipos de escalabilidade.

3.1.1 Escalabilidade Temporal no H.264/SVC

Nos padrões H.262/MPEG-2 e MPEG-4, a escalabilidade temporal é implementada basicamente segmentando-se os diferentes tipos de quadros que compõem um formato típico “IBBP” (NARVEKAR et al., 2009).

Neste contexto, de uma forma simplificada, três camadas de exibição temporal são suportadas: i) camada formada apenas por quadros I (compondo a camada base); ii) camada que contém os quadros P e iii) camada composta pelos quadros B. Esta estratégia, apesar de plenamente funcional, registra, na prática, um baixo desempenho, principalmente quando se pretende trabalhar com uma relação temporal diádica (a estrutura de predição do formato “IBBP” nem sempre está alinhada temporalmente com um formato de decomposição 2:1). Além disso, o tamanho desproporcional entre os diferentes tipos de quadros I, P e B, também acaba restringindo a flexibilidade na montagem das diferentes camadas.

Na especificação H.264/SVC foi prevista uma estrutura diferenciada de organização e orientação dos quadros de referência de cada seqüência de vídeo processada, baseando-se em uma árvore de decomposição temporal hierárquica. Além disso, foi proposta uma etapa adicional de pré-processamento visando melhor adequar as informações de vídeo a esta nova estrutura (YANG, 2007).

3.1.1.1 Estrutura B Hierárquica do SVC

Comparada com a estrutura tradicional “IBBP”, a nova organização proposta registra uma melhora significativa na eficiência de codificação, especialmente para seqüências com movimentação regular (SCHAFER et al, 2005).

Na Figura 18, pode-se observar de forma genérica como funciona esta nova estrutura. As imagens da primeira camada são chamadas de quadros chave, pois são utilizadas como referência para toda a árvore de predição. Estas imagens (quadros mais escuros) compõem a primeira camada de decomposição temporal, ou melhor, a camada base. Na prática o quadro chave representa, mais especificamente, o último quadro de um grupo de imagens (ou GOP - *Group of Pictures*).

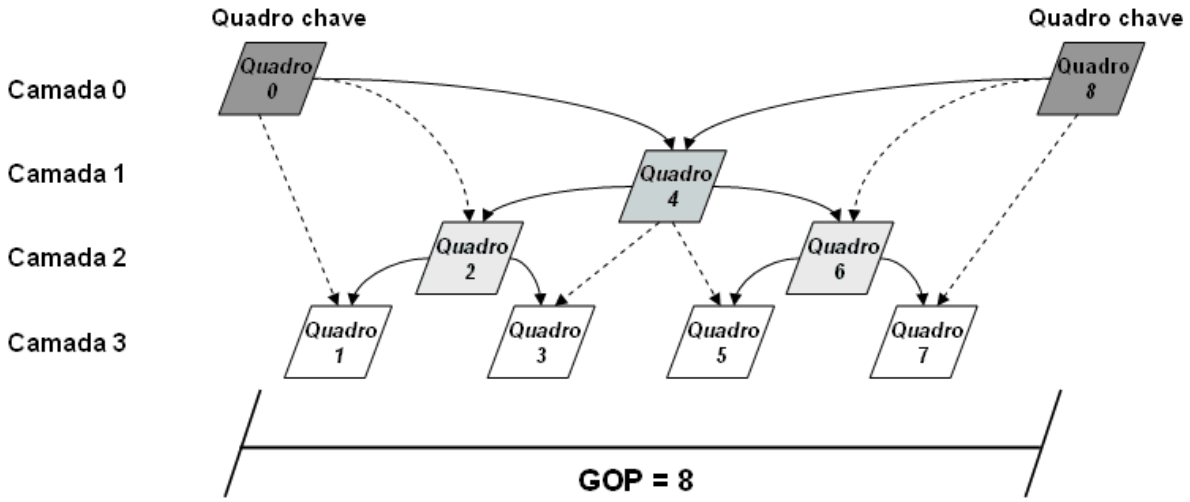


Figura 18: Formato de decomposição B hierárquico do padrão SVC

No exemplo apresentado, o tamanho do GOP é oito, sendo formado pelos quadros referenciados de 1 até 8. A segunda camada da decomposição temporal é formada pelos quadros que são imediatamente dependentes dos quadros chaves (no exemplo apresentado seria o quadro 4).

É característica deste formato que estes quadros estejam sempre situados no centro do GOP.

A próxima camada é formada pelos quadros que dependem diretamente dos quadros da segunda camada e opcionalmente dos quadros chaves (o que, no exemplo, é representada pelos quadros 2 e 6). Este procedimento de predição em árvore se repete continuamente sempre duplicando o número de quadros a cada camada.

Com exceção dos quadros chave, todos os demais quadros são do tipo B e, portanto, podem ser preditos por mais de um quadro de referência.

Um caso de bipredição é indicado na figura, onde a segunda predição é obtida pelas setas tracejadas. Devido ao seu formato de decomposição em árvore diádica (2:1) esta nova estrutura de decomposição temporal é chamada de estrutura de predição “B hierárquica” ou “pirâmide B”.

Pode-se observar pela análise da figura que o uso de uma estrutura de predição “B hierárquica” traz a limitação de operar sempre com GOPs de tamanho iguais a 2^{n-1}, onde n é o número de camadas.

Genericamente quando se trabalha com codificação escalável temporal cada camada é referenciada como *Tn*, sendo *n* o número da respectiva camada.

No exemplo da Figura 18 é apresentado o procedimento de escalabilidade temporal com quatro camadas hierárquicas (GOP=8), identificadas com diferentes tons de cinza para maior destaques.

Além das dependências entre camadas observadas na figura, é permitida adicionalmente a predição entre quadros chaves. Esta predição é possível, pois os quadros chaves podem ser quadros do tipo I ou P.

Quando um quadro chave for do tipo P este deve necessariamente se referenciar ao último quadro I ou P (no caso último quadro chave). Esta relação não foi indicada na figura apenas para simplificar o desenho da estrutura hierárquica.

A questão crucial sobre esta estrutura de predição é que quadros de uma mesma camada temporal *n* somente dependem dos quadros da mesma camada ou de camadas inferiores.

Desta forma, quando uma dada camada estiver sendo montada, os quadros com identificadores superiores a *n* podem ser descartados do fluxo saída gerado. Posteriormente estes quadros, que haviam sido descartados, serão utilizados em camadas de enriquecimento superiores.

Devido ao processo de dependência de dados, no processo de codificação primeiramente as imagens das camadas mais baixas precisam ser processadas. Somente após isso as imagens de camadas superiores podem ser produzidas, já que precisam se referenciar às imagens reconstruídas (resultado da decodificação das camadas mais baixas).

Analogamente, o mesmo processo de ordenação deve ser observado no processo de decodificação.

Assim sendo, existe uma clara diferença entre a ordem de exibição e a ordem dos processos de codificação e/ou decodificação.

Esta organização pode ser identificada na Figura 19.

Na parte inferior da figura esta relação entre ordem de codificação/decodificação e de exibição está apresentada para o caso de uma seqüência de 16 imagens transmitidas com GOP igual a oito.

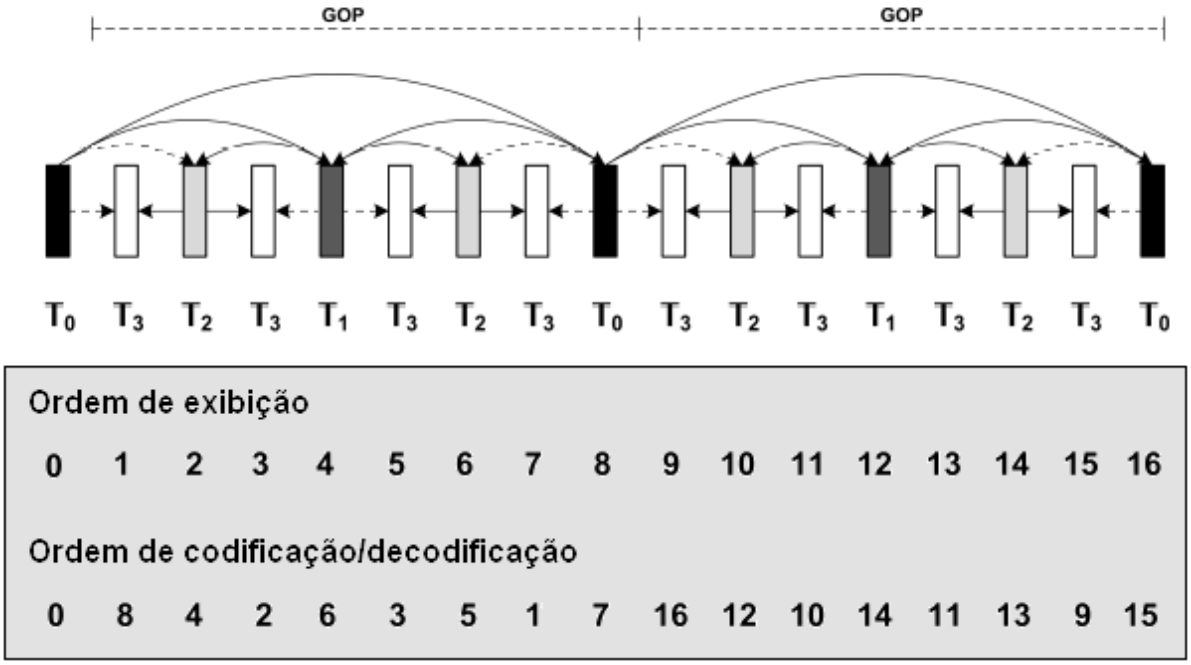


Figura 19: Ordenação de exibição e codificação em uma estrutura B hierárquica

Conforme já comentado, a estrutura de predição apresentada na Figura 19 foi idealizada visando alinhamento dos procedimentos de predição com uma relação de codificação temporal diádica, de forma a aumentar a eficiência global do codificador. A especificação de escalabilidade temporal do padrão SVC, entretanto, não se limita apenas ao modelo diádico, suportando, na realidade, a geração de camadas escaláveis com diferentes relações.

Esta maior flexibilidade somente é possível, pois os quadros do tipo B, em uma codificação de vídeo H.264, podem utilizar-se de múltiplos quadros de referência, o que também foi também herdado pelo padrão SVC. Considerando-se esta flexibilidade o padrão SVC permite a utilização de uma topologia distinta de predição de quadros, que pode ser escolhida visando melhor se alinhar com a relação desejada entre camadas temporais (SCHWARZ; MARPE; WIEGAND, 2006).

Um exemplo disso pode ser observado na Figura 20, onde se adota uma relação entre camadas não diádica.

No exemplo, o número de quadros T2 (segunda camada de enriquecimento) é seis vezes maior que o número de quadros T1 (primeira camada de enriquecimento).

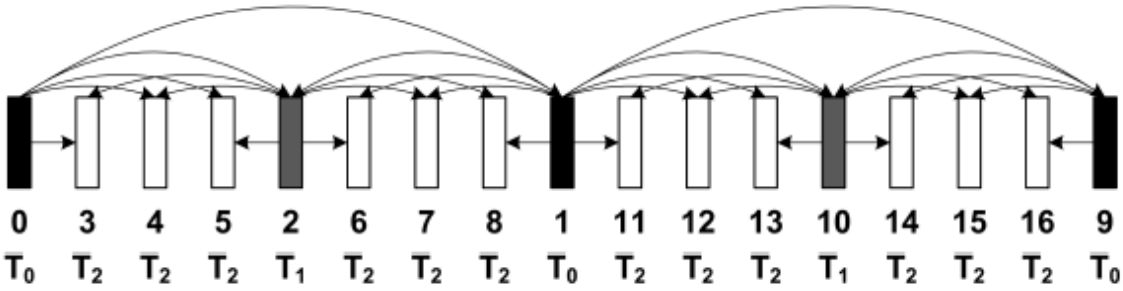


Figura 20: Codificação temporal não diádica usando a estrutura B hierárquica

(RIECKL, 2008)

Os modelos apresentados utilizam-se da flexibilidade do modelo de predição do H.264 a fim de aumentar a eficiência do codificador. Estes métodos, entretanto trazem como desvantagem o aumento da latência de codificação. Considerando-se o caso da Figura 19 pode-se observar que a discrepância entre a ordem de codificação e exibição chega a uma distância de oito quadros.

De forma genérica, pode-se dizer que este modelo demonstrado chega a causar um tempo de latência igual a um GOP inteiro para codificação. Dependendo do tipo de aplicação pretendida este atraso pode ser proibitivo.

Devido à grande flexibilidade do modelo de predição do padrão H.264, porém, é possível criarem-se soluções que reduzam o tempo de latência do codificador para valores menores, tais como GOP/2 ou GOP/4 (RIECKL, 2008).

No caso mais extremo é possível inclusive chegar-se a uma solução, ainda condicionada ao formato hierárquico, onde a predição de movimento não possui atraso estrutural no processo de codificação.

Esta solução é apresentada na Figura 21 onde se pode perceber a eliminação da predição de imagens do futuro. Esta solução também reduz a complexidade computacional do codificador, bem como os requisitos de memória.

Deve-se, entretanto saber que a solução de atraso zero apresentada na figura, exatamente por ter sacrificado a utilização de quadros futuros como referência de predição, terá uma eficiência de codificação inferior à registrada para os demais casos apresentados.

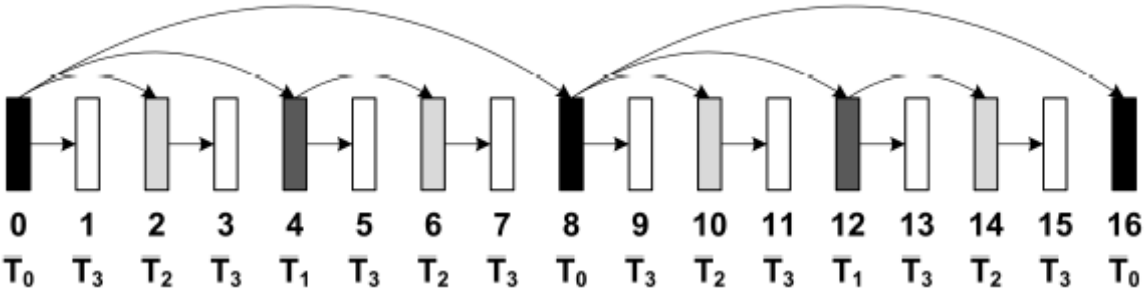


Figura 21: Modelo de codificação temporal sem atraso de predição

(RIECKL, 2008)

Para fins de validação, o grupo JVT realizou diversos experimentos onde foram analisadas estruturas B hierárquicas com e sem qualquer restrição de atraso (Figuras 19 e 21) foram comparadas com a mesma codificação utilizando uma estrutura "IBBP" tradicional (SCHAFER, 2005). Para maioria das seqüências observadas a nova estrutura proposta apresentou ganhos na eficiência de codificação, mesmo para a topologia sem atrasos de latência. A partir destes experimentos pode-se deduzir que prover escalabilidade temporal pela estrutura B hierárquica do SVC não tem impacto negativo na eficiência de codificação.

3.1.1.2 Filtragem Temporal Compensada em Movimento

Durante o desenvolvimento do padrão SVC, percebeu-se que muitas vezes as informações obtidas pelos mecanismos tradicionais de predição H.264 se mostravam limitadas, principalmente quando se realiza a segmentação do vídeo em diversas camadas temporais.

Assim sendo, foi sugerido um algoritmo de filtragem adicional que, atuando como uma ferramenta de pré-processamento ao módulo de predição de movimento, que buscasse aumentar a eficiência de codificação (ZHANG; ZAFAR, 1992).

O algoritmo sugerido recebeu o nome de MCTF (*Motion-Compensated Temporal Filtering*) e se baseia na decomposição e reconstrução da imagem ao longo da trajetória do movimento, aumentando assim a correlação entre as camadas filtradas (SCHWARZ; MARPE; WIEGAND, 2006).

A Figura 22 apresenta um exemplo simplificado de um filtro MCTF de dois *taps*. Seu princípio de funcionamento se baseia em três passos: operação polifásica, predição e atualização. A operação polifásica decompõe as imagens originais em dois conjuntos de imagens: com índices pares (2k) e impares (2k+1). O módulo P representa o passo de predição (*prediction*) e U o passo de atualização (*update*). A imagem de entrada (sinal *Sk*), após passar pelo processo de decomposição temporal polifásico (no caso 2:1) é separada em dois conjuntos de imagens: as de índice par (*2k*) e as de índice ímpar (*2k+1*). Concluído o processo de decomposição (que compreende os passos de predição e atualização) são geradas duas imagens de saída: uma resultante de uma filtragem passa-alta (*Hk*) e outra resultante de uma filtragem passa-baixa (*Lk*). A imagem reconstruída, a partir da operação inversa, é identificada com *Sk'*(SCHAFER et al, 2005).

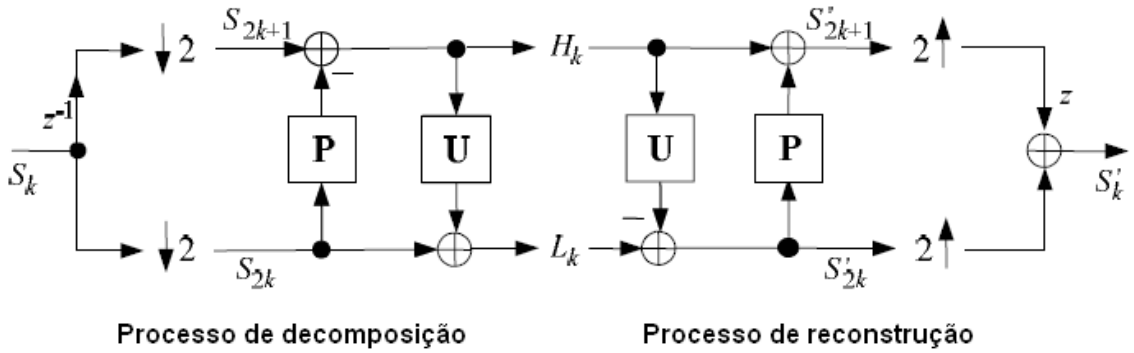


Figura 22: Exemplo resumido de um filtro MCTF de dois taps

(SCHAFER et al, 2005)

O processo de decomposição temporal MCTF segue também uma estrutura hierárquica, onde as imagens de índice ímpar são preditas (passo P) de imagens adjacentes de índice par, produzindo imagens do tipo passa-alta. As imagens de índice impar são atualizadas (passo U) para gerar imagens do tipo passa-baixa, em combinação com as imagens passa-alta mais próximas.

Para prover a escalabilidade temporal de múltiplos níveis, a decomposição precisa ser realizada recursivamente sobre as imagens passa-baixa anteriormente produzidas durante o processamento de cada uma das diferentes camadas.

Uma representação deste procedimento é apresentada na Figura 23. O exemplo, apresentado ilustra o princípio de decomposição temporal baseado na técnica MCTF, realizada sobre uma seqüência de 16 quadros. As setas que geram as imagens H (passa-alta)

são as que representam o passo de predição (P), enquanto que as setas que geram as imagens L (passa-baixa) representam a etapa de atualização (U).

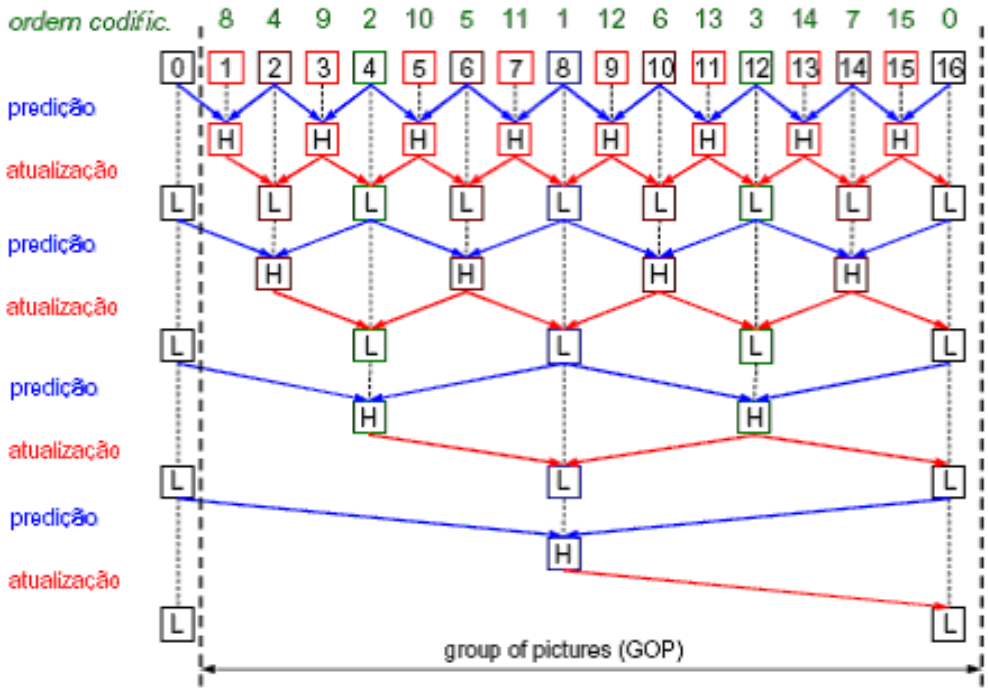


Figura 23: Princípio de filtragem completa MCTF (decomposição)

(SCHAFER et al, 2005)

Pode-se perceber que, assim como o caso da estrutura B hierárquica, também o processo de decomposição temporal MCTF se apresenta nativamente alinhado com a escalabilidade temporal diádica. As imagens passa-baixa finais estão relacionadas com a camada base, enquanto que as imagens passa-alta obtidas pelos demais passos intermediários, estão relacionadas com as diferentes camadas de enriquecimento.

Para realizar os procedimentos de decomposição e reconstrução, a técnica MCTF utiliza transformadas Wavelets do tipo Haar ou 5/3. Cada uma destas transformadas traz padronizados os passos de predição (P) e atualização (U). Particularmente com o uso de transformadas Wavelets do tipo 5/3 deve-se realizar uma predição bidirecional completa para a decomposição temporal. Quando a transformada Haar é selecionada, por suas próprias características de simetria, apenas uma predição do tipo unidirecional é necessária. Neste caso, alguns caminhos de predição e atualização (Figura 23) podem ser removidos. A predição unidirecional pode ser apenas referente a quadros passados ou referente a quadros futuros (RIECKLER, 2008).

A seleção de uso de predição unidirecional ou bidirecional (ou seja, entre utilizar-se transformada do tipo Haar e 5/3 respectivamente) pode ser feita de forma adaptativa para cada bloco (OHM, 2005). Pode-se perceber, pela própria estruturação em árvore do algoritmo MCTF, que uma decomposição completa causa um atraso de latência de um GOP inteiro. Além disso, a implementação deste mecanismo gera uma grande demanda de memória para guardar tantos quadros de referência. Conforme o caso estas características podem tornar o processo de codificação impraticável para algumas aplicações. Visando reduzir estes problemas alguns passos de predição e atualização referentes a quadros futuros, foram removidos do projeto original (SCHWARZ; MARPE; WIEGAND, 2006).

A remoção destes passos de predição e atualização produziu uma redução dos requisitos de memória e do atraso de codificação até em cerca de quatro vezes. A Figura 24 apresenta esta nova estrutura, considerando-se o mesmo caso de uma seqüência de 16 quadros, onde se percebe um atraso de latência igual a um quarto do tamanho do GOP.

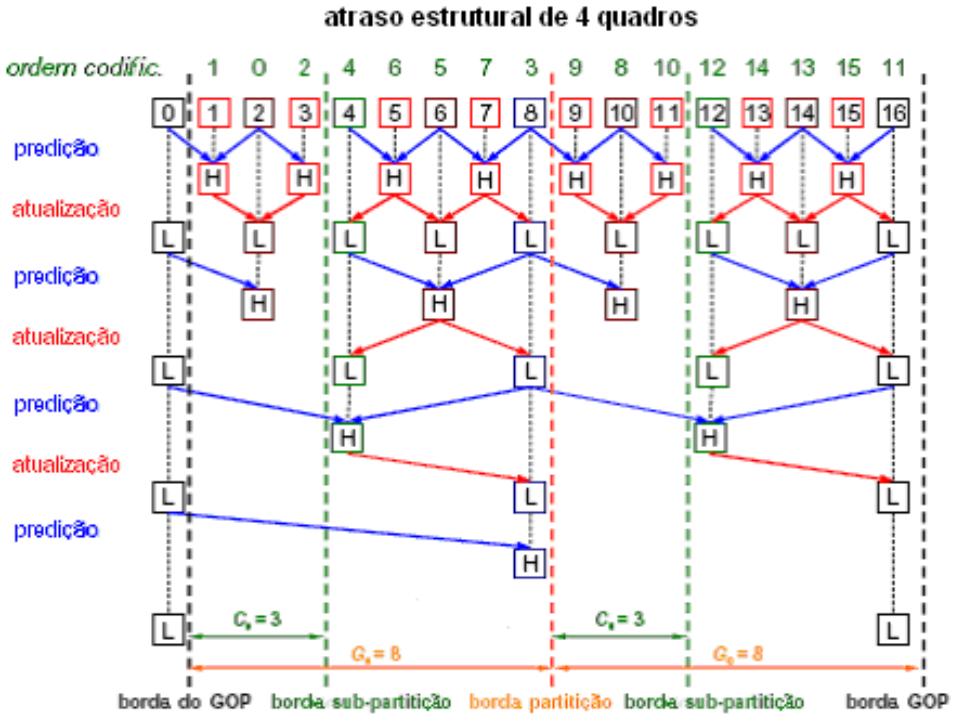


Figura 24: Princípio de filtragem MCTF simplificada para menor atraso

A complexidade do mecanismo MCTF, mesmo em sua versão de menor atraso, acaba sendo significativa devido à necessidade de se utilizar transformadas Wavelet, o que pode comprometer algumas implementações para dispositivos de baixo poder de processamento. Com base nestes fatos, o mecanismo MCTF foi excluído como etapa obrigatória do padrão SVC, podendo ser utilizado como recurso adicional conforme disponibilidade de recursos.

3.1.2 Escalabilidade Espacial no SVC

O processo de escalabilidade espacial sobre codificadores de vídeo trata da montagem de diferentes camadas complementares cada uma dela operando com uma resolução distinta (ex: QCIF, CIF e 4CIF).

Para uma codificação multiresolução (multicamadas) o padrão SVC adota uma estrutura de decomposição em pirâmide espacial (similar às abordagens MPEG-2 e MPEG-4). Nesta estrutura as imagens de maior resolução são convertidas em imagens de resoluções mais baixas por um procedimento de subamostragem baseado em filtragem ou decimação. O processo de subamostragem se repete recursivamente para cada camada a ser gerada, desde a imagem maior resolução até a camada de menor resolução (camada base). Após a conclusão do procedimento de subamostragem pode-se iniciar o processo de codificação.

Primeiramente, as imagens de baixa resolução são codificadas. As imagens de resoluções mais altas só podem ser codificadas depois, uma vez que estas devem se referenciar às mais baixas.

Para ilustrar este procedimento, a Figura 25 apresenta o procedimento genérico adotado para implementar uma aplicação de escalabilidade espacial de quatro camadas no padrão H.264/SVC.

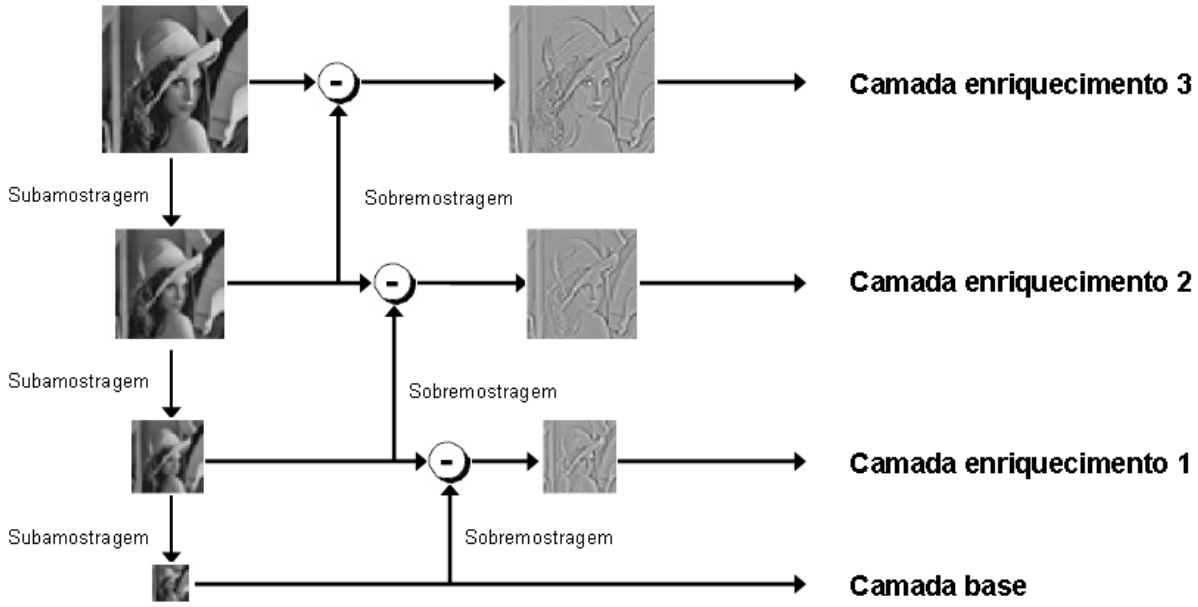


Figura 25: Mecanismo de predição espacial do padrão H.264/SVC

O vídeo de entrada, apresentado na parte superior da figura, representa o vídeo de maior resolução. Cada camada inferior opera com um vídeo de menor resolução. Para tanto o vídeo original passa por vários procedimentos de sub-amostragem em sequencia, até se chegar na resolução da camada base. O codificador da camada base então comprime este vídeo de forma autônoma (sem se referenciar aos demais). Ao final desta operação, o vídeo recuperado pela camada base, será usado como referência para a camada de enriquecimento imediatamente superior. A idéia é que esta camada de enriquecimento não opere diretamente sobre um vídeo de maior resolução, mas sim somente sobre os dados complementares em relação à camada base. Sendo assim o vídeo reconstruído na camada base passa por um processo de sobreamostragem. O resultado deve ser subtraído do vídeo da camada 1, gerando os resíduos necessários para uma conversão da camada base para esta nova camada. Estes resíduos passam por um processo de codificação completo, gerando as informações da camada de enriquecimento 1. Os dados recuperados pela camada 1 serão usados como referência para a camada 2. De forma similar, os procedimentos então se repetem sistematicamente até a última camada (SEGALL; SULLIVAN, 2007).

O perfil mais simples no padrão SVC é chamado de perfil *Baseline* (*Scalable Baseline Profile*). O perfil *Baseline* é voltado para implementações com menores resoluções de tela (SD). Suporta apenas razões entre resoluções iguais a 1,5 ou 2 e deslocamentos na tela com resolução inteira de macroblocos, ou seja os deslocamentos devem ter valores múltiplos de 16 pixels. Cada camada é codificada independentemente das camadas superiores, podendo, entretanto valer-se de recursos de predição temporal e de movimento utilizando-se imagens de referência.

Visando maior flexibilidade de aplicação, o padrão SVC permite que se utilize qualquer relação de resoluções entre duas camadas consecutivas, não se limitando ao caso clássico diádico. Na seleção da área a ser utilizada para montar a camada base são permitidas operações variáveis de corte (definição de altura e largura da área a ser utilizada para a próxima camada) e deslocamento (ponto inicial onde o corte será feito) sobre a imagem de maior resolução. É possível inclusive utilizar aspectos de resolução distintos entre duas camadas (exemplo imagem de resolução de aspecto 16:9 para 4:3).

A Figura 26 ilustra genericamente os parâmetros de configuração para montagem de uma camada base, a partir de uma imagem de maior resolução.

Pode-se perceber que a camada base consegue referenciar seu ponto inicial (x_{orig}, y_{orig}) a partir de qualquer posição da imagem original. Os valores de w_{extrac} e h_{extrac} definem as

dimensões de largura e altura, respectivamente, da região de corte que servem para formar a imagem da camada base. O procedimento de subamostragem indicado é responsável por converter a imagem cortada para as dimensões finais da camada base (w_{base} e h_{base} respectivamente) (SEGALL; SULLIVAN, 2007).

O caso genérico, apresentado na figura, que não se restringe à relação diádica (razão entre camadas igual a dois) é chamado de escalabilidade espacial estendida ou ESS (*Extended Spatial Scalability*).

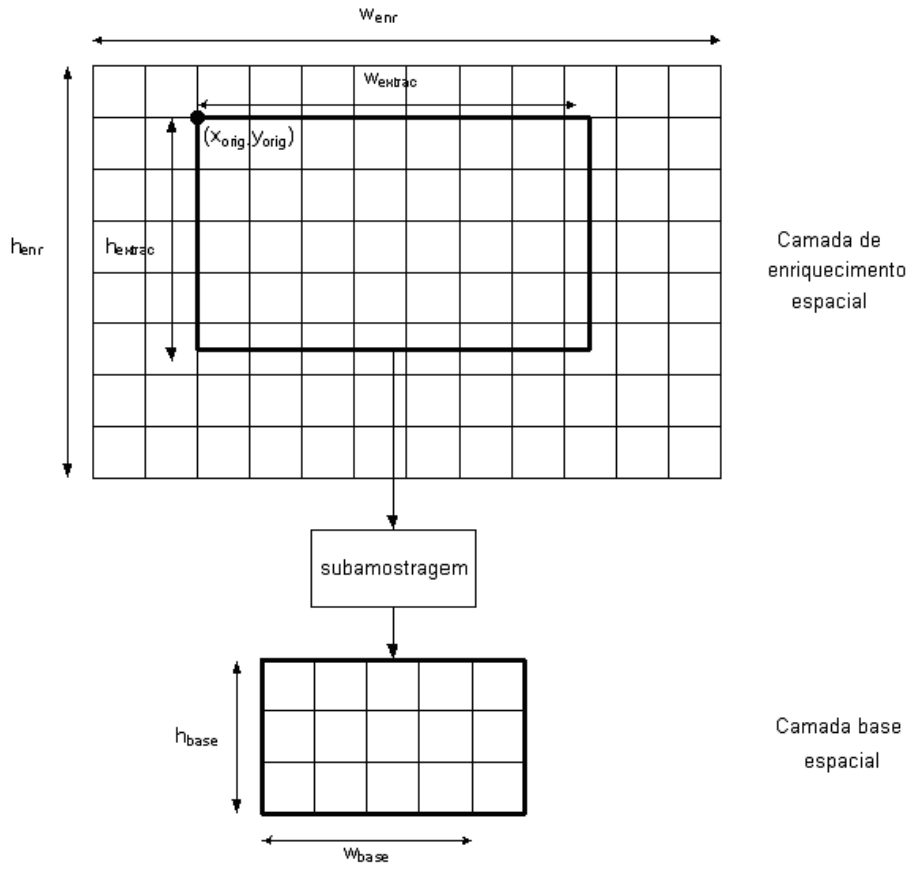


Figura 26: Parâmetros de relação da camada base com a de enriquecimento (SEGALL; SULLIVAN, 2007)

Os perfis escaláveis dos padrões MPEG-2 e MPEG-4 implementam a predição utilizando como referência a imagem reconstruída da camada mais recente. Em ambos os perfis, primeiramente a imagem de baixa resolução precisar ser completamente decodificada. Na seqüência ela é sobreamostrada e a partir daí pode passar por procedimentos de predição entre camadas. Este procedimento seqüencial de decodificação, sobreamostragem e predição deve se repetir para cada nova camada, gerando-se o que se chama de decodificação em múltiplos laços.

A decodificação em múltiplos laços de compensação de movimento tem maior eficiência, porém aumenta consideravelmente a complexidade do algoritmo, bem como o consumo de memórias de referência, além de gerar indesejáveis dependências seqüenciais. Uma solução mais simples é adotada pelo padrão SVC.

A nova solução proposta para o SVC não trabalha com diferentes resoluções intermediárias, mas considera sim sempre apenas a resolução final da imagem (ou seja, a resolução a ser utilizada na exibição) independente da resolução da camada. Ou seja, a imagem é sempre expandida para a resolução da exibição final, independentemente do estágio em que se encontre. Esta solução é chamada de decodificação de laço único. A decodificação de laço único reduz ligeiramente a eficiência de codificação, mas simplifica significativamente a estrutura do decodificador (SEGALL; SULLIVAN, 2007).

Segundo proposto, a sobreamostragem dos blocos de luminância para uma resolução mais alta é feita pela aplicação de um filtro de interpolação polifásico de quatro *taps*. Já os componentes de crominância são sobreamostrados por um núcleo de interpolação bilinear (SCHAFER et al, 2005).

Um dos recursos mais importantes para garantir alta eficiência do codificador SVC é o mecanismo de predição. Considerando-se a importância deste mecanismo para remover redundância de informações foi desenvolvido um módulo aprimorado denominado de módulo de predição entre camadas ou ILP (*Inter-Layer Prediction*).

A predição entre camadas é flexível por prever diferentes relações entre os diferentes tipos de camadas usadas, sempre porém baseando a precisão do módulo em aritmética inteira. Neste sentido, para reduzir sua complexidade, as fórmulas podem ser desenvolvidas utilizando-se operações em complemento de dois com resolução de 16 bits (neste caso adotando precisão igual a 16).

As multiplicações e divisões adotadas pelo algoritmo são potência de dois, podendo assim ser facilmente implementadas por simples deslocamentos de bits. Os arredondamentos são feitos também por operações simples como soma (metade do valor de prioridade) e deslocamento à direita.

Como forma de aumentar a eficiência de codificação o padrão SVC não se limita à predição de informações de movimento. De fato, o padrão SVC prevê três técnicas de predição entre camadas: predição de movimento, de textura intra e de resíduo.

3.1.2.1 Predição de Movimento Entre Camadas

A predição de movimento é o método mais comum usado para remover redundâncias de informações. Neste método, os vetores de movimento provenientes de camadas mais baixas podem ser usados pelas camadas de enriquecimento superiores (NARVEKAR et al., 2009).

Adicionalmente aos modos de macrobloco disponíveis no padrão H.264/AVC, o SVC criou o modo de camada base (*baselayer mode*), que é usado para predição de movimento entre camadas. O modo de camada base reusa a informação de movimento da camada de referência sem gastar bits extras. Se este modo não for selecionado uma informação de movimento independente será calculada.

Quando o recurso de predição de movimento entre camadas for utilizado é importante incluir junto aos vetores de movimento propriamente ditos, dados como índice da imagem de referência e tipo de partição de macrobloco, uma vez que são suportados modos ou partições tipo 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 e 4x4 (MPEG, 2008).

Os dados dos vetores de movimento da camada inferior, bem como as diferentes partições, devem ser reescalonados para serem adequadamente interpretados pelas camadas mais altas. Um exemplo disso pode ser observado na Figura 27. No exemplo, a relação entre camadas é diádica, podendo-se perceber com maior clareza a transposição das informações de movimento de uma camada para outra.

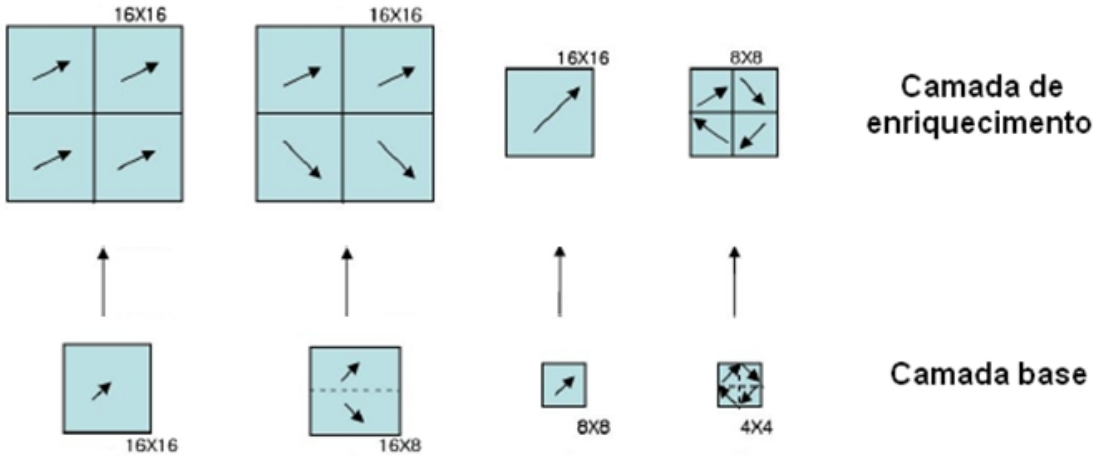


Figura 27: Aproveitamento de informações de movimento da camada base

Adaptado de (MPEG, 2008)

3.1.2.2 Predição de Textura Intra

Como já comentado, a predição de movimento quando aplicada para múltiplas camadas espaciais pode levar a uma complexidade significativa. A predição de textura intra é uma alternativa importante prevista para reduzir a complexidade dos algoritmos na busca por redundância de informações. Este recurso proposto para o padrão H.264 permite a predição de textura intra, somente considerando blocos internos da mesma camada de referência. O bloco intra predito na camada de referência pode ser usado por outros blocos intra em camadas superiores.

A norma SVC prevê o uso simultâneo de predição de movimento entre camadas e textura intra. Se, entretanto, todos os blocos de luminância 4x4 do macrobloco de enriquecimento forem obtidos sem movimentação, o macrobloco é referenciado de forma especial, neste caso, sendo é denominado como macrobloco do tipo “I_BL”.

Um exemplo desta situação é apresentado na Figura 28, onde se observa que não há informação de movimento no processo, apenas sobreamostragem da camada inferior para atingir a mesma resolução da camada superior e uma operação de cálculo do erro envolvido entre as duas texturas.

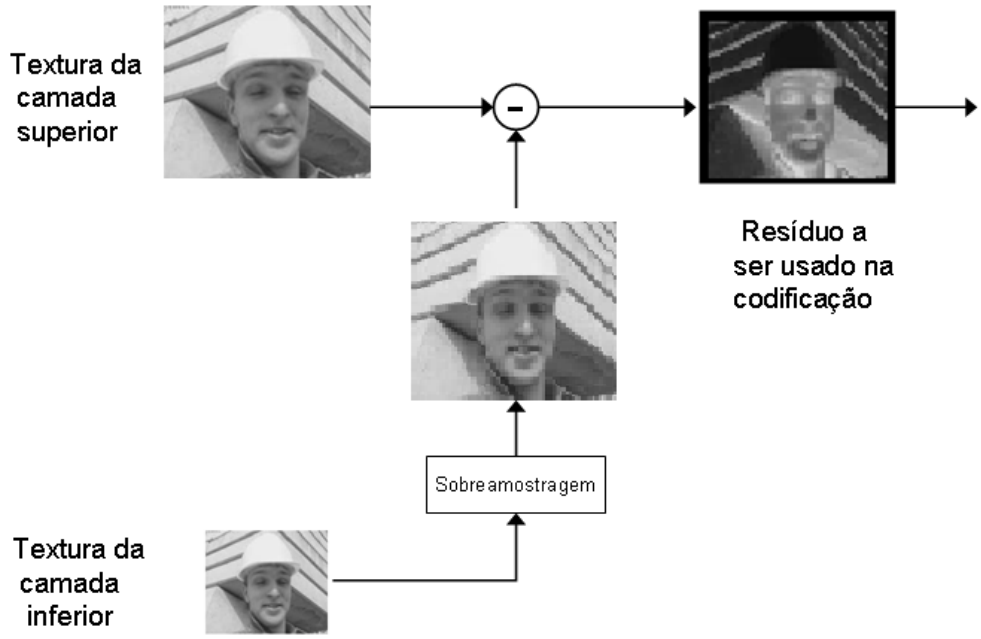


Figura 28: Predição de textura intra

3.1.2.3 Predição de Resíduos entre Camadas

Um terceiro tipo de predição é usado para reduzir a energia de resíduos após a predição temporal. No padrão H.264/SVC esta predição, chamada de predição de resíduo da compensação de movimento é realizada no domínio espacial.

Este recurso adicional baseia-se no fato de que quando se determinam informações de movimento similares entre camadas (resultado do processo de predição de movimento), os resíduos obtidos entre camadas consecutivas também devem apresentar alta correlação.

Em alguns casos é possível que camadas consecutivas tenham movimentos independentes, e nestas condições os resíduos de duas camadas consecutivas podem não estar correlacionados. Desta forma, o recurso de predição de resíduos, a critério do codificador, pode ser ou não usada, de forma adaptativa, em nível de macrobloco.

De forma prática este mecanismo de predição é bastante similar ao mecanismo de predição de textura intra, com a diferença que neste caso os blocos utilizados são resultantes do cálculo de resíduo (após a realização da compensação de movimento). A Figura 29 apresenta de forma análoga o procedimento de predição de resíduo para o exemplo anterior.

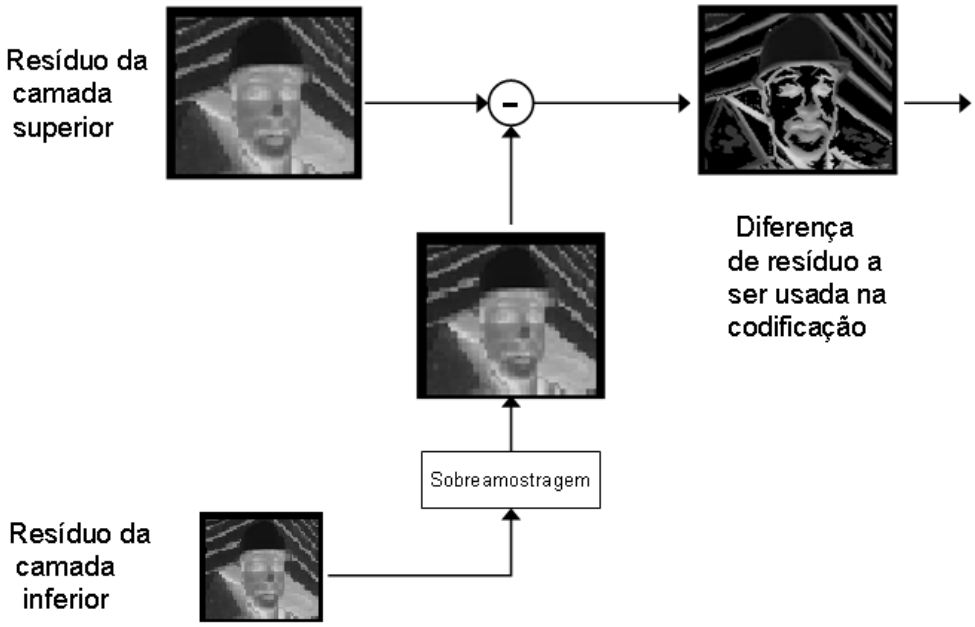


Figura 29: Predição de resíduo entre camadas

Quando a predição de resíduos é realizada, ao invés de se decodificar todas as amostras da camada base, apenas as informações de resíduo são compensadas para a camada mais alta. Na prática, os sinais de resíduo recebidos da camada base são sobreamostrados

utilizando-se um filtro bilinear sobre os blocos de informação. É necessário adicionalmente filtrar as bordas destes blocos a fim de evitar que estas causem distorções visuais sobre o vídeo reconstruído.

3.1.3 Escalabilidade SNR no SVC

O mecanismo de escalabilidade SNR é implementado no domínio das freqüências, ou seja, após as informações de textura e resíduo passarem pelo procedimento de transformada (DCT). De forma genérica, este tipo de escalabilidade se baseia na segmentação dos valores de quantização adotados para cada camada, produzindo desta forma diferentes níveis de qualidade. Cada camada, neste caso, codifica apenas as informações complementares obtidas em seu nível de qualidade, o qual está relacionado com o valor adotado de QP (*Quantization Parameter*), em relação à camada imediatamente inferior, que estará usando outro valor de QP. Este procedimento está ilustrado na Figura 30, onde quatro camadas escaláveis são geradas.

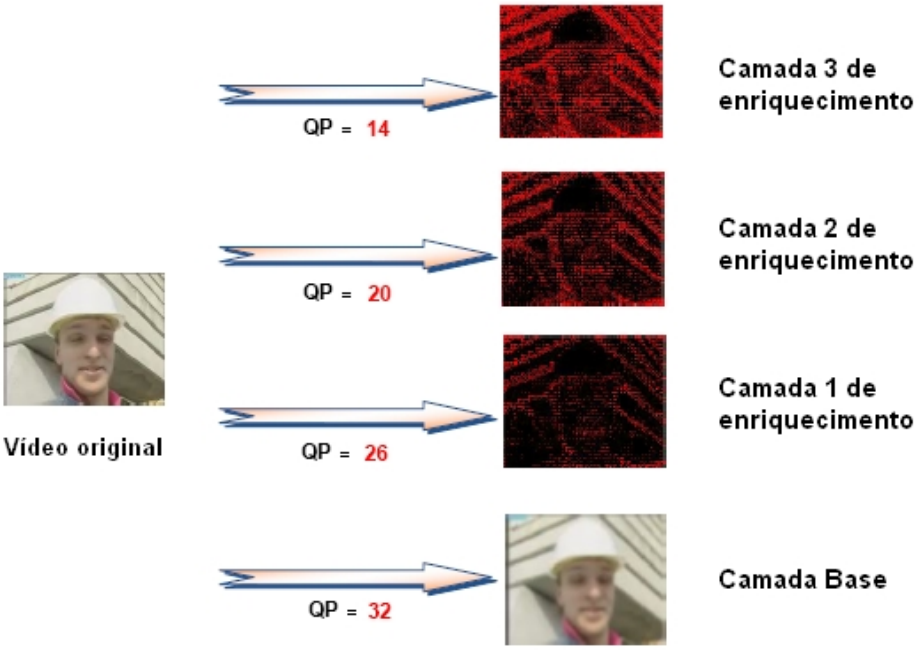


Figura 30: Exemplo de codificação escalável SNR variando-se QP

Adaptado de (MPEG, 2008)

Em claro aprimoramento sobre os perfis escaláveis anteriores, a especificação de escalabilidade SNR prevista para o padrão SVC inclui três versões distintas: CGS, MGS e FGS (RIECKL, 2008).

3.1.3.1 Versão CGS

A primeira abordagem de escalabilidade SNR prevista tem um princípio de funcionamento similar ao perfil escalável SNR do padrão MPEG-2. Nesta abordagem o codificador realiza seu procedimento de predição de movimento considerando informações das camadas base e de enriquecimento. O codificador MPEG-2 pode usar ambas as camadas ou apenas a camada base no laço de predição. Esta abordagem provê alta eficiência quando ambas as camadas são recebidas, mas pode gerar o fenômeno de *drift*[3], que ocorre quando apenas a camada base é recebida, o que causa falha na obtenção de informações de camadas superiores.

A fim de reduzir o problema do *drift*, o padrão SVC aprimora o mecanismo proposto pelo MPEG-2 gerando uma versão denominada como CGS (*Coarse Grain Scalability*). Na versão CGS do padrão SVC, cada camada implementa seus procedimentos de predição de forma independente, ou seja, as imagens de referência de cada camada devem pertencer ao mesmo nível de qualidade (COCK; NOTEBAERT; WALLE, 2007). Esta solução resolve o problema do *drift* ao mesmo tempo em que otimiza a extração de informações para cada camada.

Alguns autores consideram a especificação CGS como um caso especial de escalabilidade espacial, pois explora todos os seus recursos de predição. Neste caso, apenas o processo de sobreamostragem é desnecessário uma vez que todas as camadas envolvidas têm a mesma resolução (HUANG; PENG; CHIANG, 2007).

A principal desvantagem da versão CGS é que apresenta um número reduzido de níveis de qualidade distintos, e por isso é referenciado como um mecanismo de granularidade grosseira (*coarse granularity*).

3.1.3.2 Versão MGS

Como segunda alternativa, o padrão SVC introduz uma especificação adicional de escalabilidade SNR criada visando melhorar a granularidade da codificação escalável, ao

[3] O fenômeno denominado *drift* define a situação em que os laços de predição de movimento no codificador e decodificador não estão trabalhando sobre as mesmas imagens de referência, ou seja, estão fora de sincronismo.

mesmo tempo em que mantém o *drift* em um nível aceitável. A técnica é chamada de escalabilidade de granularidade média ou MGS (*Medium Grain Scalability*). Os aprimoramentos sugeridos basicamente dizem respeito à predição de movimento.

Nesta nova abordagem a predição de movimento pode ser conduzida pelas camadas base e de enriquecimento, assim como acontecia na versão escalável MPEG-2.

Para se reduzir o problema do *drift*, a especificação MGS prevê a inclusão de períodos de atualização na camada base. Na prática este período de atualização ocorre sempre que surgem os chamados quadros chave. Nestes momentos, o laço de compensação de movimento do lado do decodificador, independente da situação em que se encontre, consegue se sincronizar com o codificador. Assim o padrão MGS garante que o efeito de *drift* não se torne muito agudo.

A versão MGS permite o uso de até 16 níveis de qualidade distintos (ou seja, uma camada base e até 15 camadas de enriquecimento SNR), aumentando assim a flexibilidade do codificador para diferentes de taxas de bit. Na Figura 31, apresenta-se uma comparação entre as diferentes versões de escalabilidade SNR do padrão SVC.

No exemplo da codificação MGS, o quarto quadro da seqüência representa um quadro chave, onde se pode identificar a mudança do mecanismo de predição de movimento que é usado para sincronização.

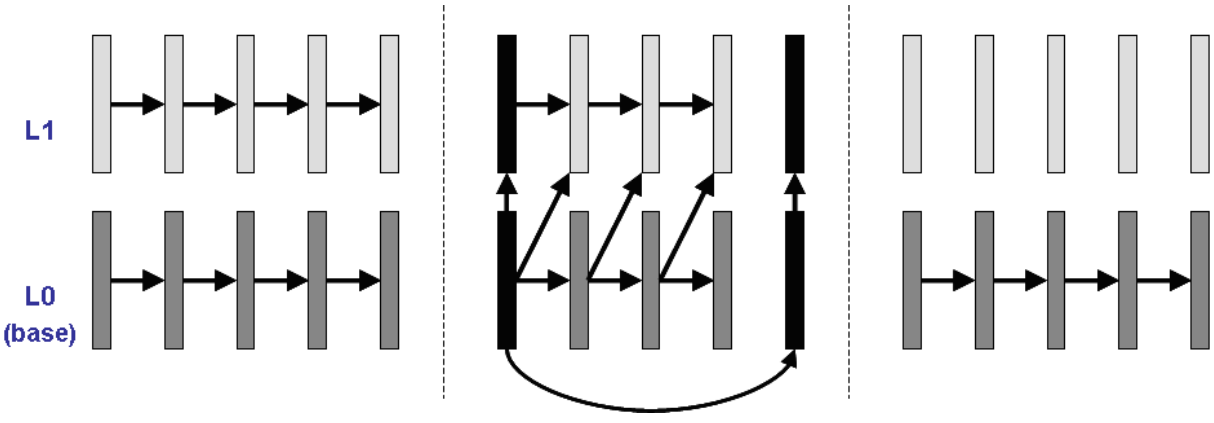


Figura 31: Comparação entre os diferentes tipos de escalabilidade SNR

(a) CGS (b) MGS e (c) FGS

3.1.3.3 Versão FGS

A especificação FGS ao contrário das duas versões anteriores de escalabilidade SNR do SVC não possui um número pré-definido de camadas como limite. Na verdade a versão FGS visa prover uma adaptação quase contínua da taxa de bit de saída em relação à banda de rede disponível. Para tanto esta versão escalável utiliza uma técnica aprimorada de codificação em planos de bit que permite a truncamento de informações em qualquer ponto arbitrário, a fim de suportar o refinamento progressivo dos coeficientes de transformada.

O método FGS realiza a predição de movimento considerando a camada de qualidade mais baixa (camada base) como imagem de referência. Uma vantagem desta estratégia é que o decodificador tem sempre a mesma qualidade de vídeo na imagem de referência, ou seja, a possível perda de pacotes de enriquecimento não influencia no laço de predição de movimento. A camada de enriquecimento é codificada apenas internamente (sem predição entre quadros consecutivos), prevenindo assim o problema de erro de *drift*. Garante-se, desta forma, que o codificador e o decodificador estejam sincronizados todo o tempo (PARK; YOO; SUH, 2006).

A maior desvantagem desta solução está na reduzida eficiência de codificação obtida, uma vez que as camadas de enriquecimento não se valem do recurso de predição de informações entre camadas (MPEG, 2008).

A Figura 32 traz um exemplo que ilustra o mecanismo de varredura em zigzag e a sinalização de bits significativos para um bloco de 4x4 pixels.

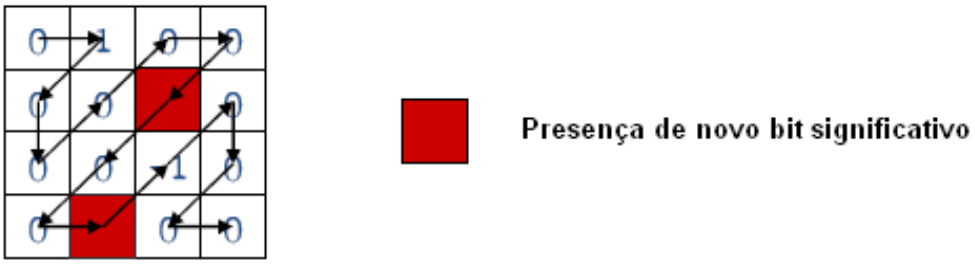


Figura 32: Mecanismo de varredura em um bloco 4x4 para escalabilidade FGS

Para cada bit significativo deve ser apresentado em uma estrutura composta por duas informações complementares: sua posição específica (índice na seqüência de varredura zigzag) e o valor de corrida (quantidade de elementos nulos entre o símbolo atual e o último

bit válido). No padrão SVC cada símbolo deste gerado será usado para alimentar dois possíveis métodos de entropia: mecanismo CABAC (sugerido originalmente pela especificação por apresentar maior compressão) ou CAVLC, mais usado para aplicações que visem reduzir complexidade e tempo de execução. O uso destes algoritmos de entropia em conjunto com o FGS pode aumentar a eficiência de codificação (HUANG; PENG; CHIANG, 2007).

3.2 VISÃO GERAL DO PADRÃO H.264/SVC

Pode-se perceber, de acordo com o apresentado nas seções anteriores, que a implementação prática de um codificador escalável SVC completo é realmente complexa, uma vez que incorpora um grande número de algoritmos distintos. A integração destes algoritmos exige um adequado planejamento, uma vez que muitos deles operam em diferentes níveis de abstração (tais como manipulação de pixels, macroblocos, informações de contexto, entre outros).

A fim de facilitar o entendimento da arquitetura genérica de um codificador de vídeo H.264/SVC, na Figura 33 se apresenta um diagrama de blocos simplificado que ilustra os seus principais módulos internos, bem como as interligações de dados entre si. No exemplo se representa um codificador de três camadas distintas, onde cada camada possui uma entidade de codificação própria. A figura apresenta a visão geral de um codificador escalável simplificado. Para um codificador mais complexo, que exija um número maior de camadas, basta-se replicar novas entidades de codificação, seguindo-se a mesma lógica de ligações apresentada.

As interligações entre os diferentes módulos podem ser relacionadas a dados de textura, vetores de movimento ou resíduos. Cada um destes tipos é discriminado na figura sobre as linhas que interligam os módulos internos do codificador. De fato, para garantir a execução e processamento correto dos algoritmos individuais, além da interligação é importante se preocupar com a sincronização de informações entre estes.

No exemplo apresentado o codificador adota escalabilidade SNR (entre camada base e camada 1) e espacial (entre camada 1 e camada 2). Como já comentado, a adoção da escalabilidade espacial leva à utilização de mecanismos de subamostragem e sobreamostragem.

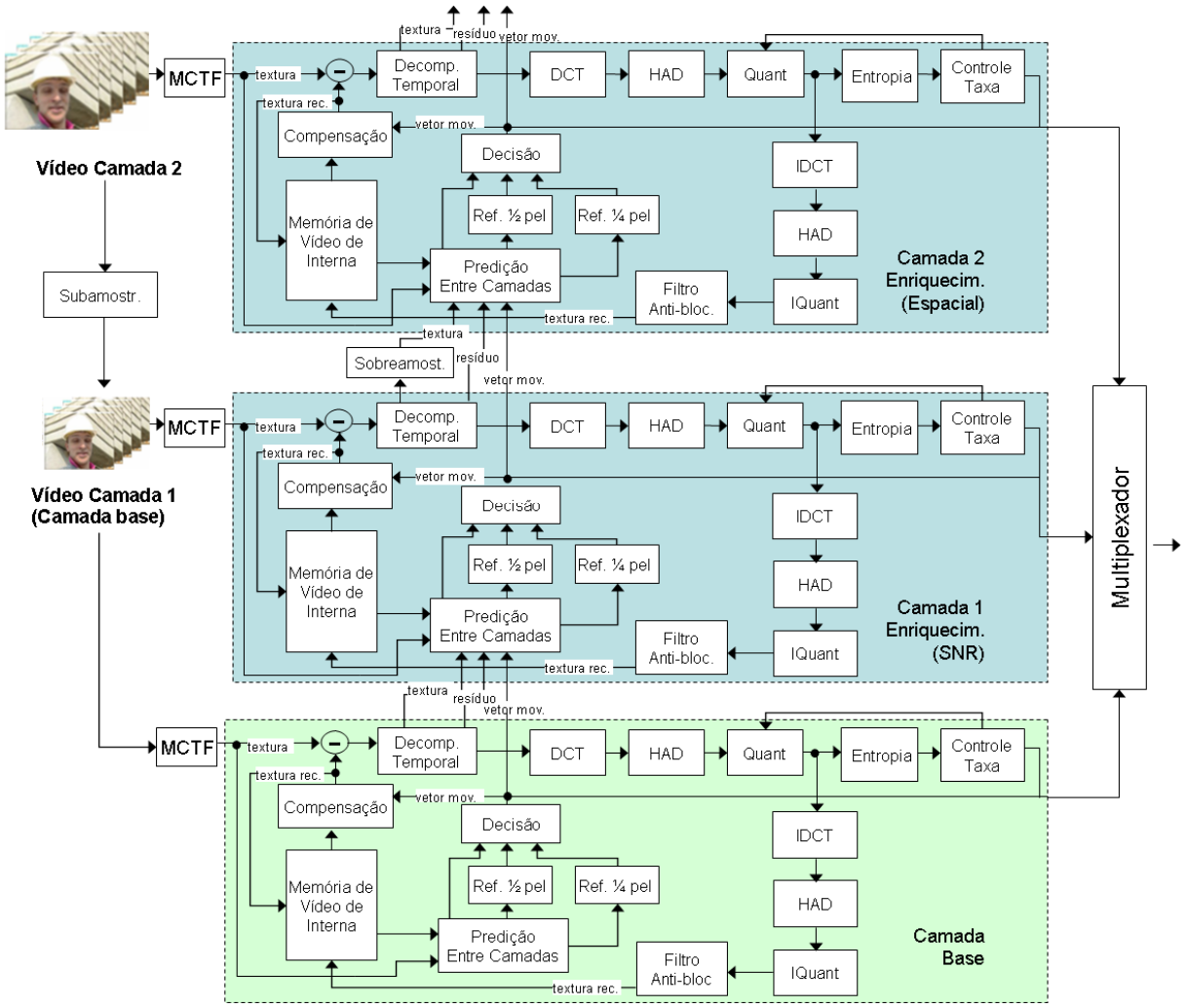


Figura 33: Diagrama de blocos simplificado de um codificador SVC

Observando-se o diagrama de blocos pode-se perceber que o primeiro módulo a trabalhar sobre as imagens de entrada é o módulo MCTF. Sua função é a de melhor adequar as informações de textura da imagem de entrada para o procedimento de decomposição temporal, a ser posteriormente implementado pelo codificador. Por se tratar de um estágio opcional, este não foi incluído como algoritmo componente de cada estrutura de codificação de camadas (indicada pela seleção em destaque).

O módulo de “Predição Entre Camadas” é um dos mecanismos mais importantes do codificador SVC. Sua função é identificar as redundâncias espaciais e temporais, gerando informações de vetores de movimento e resíduo. Em um codificador convencional este módulo se baseia em comparar a imagem de entrada com quadros de referência. Já no padrão H.264/SVC novas funcionalidades devem ser inclusas para este módulo de predição, principalmente buscando-se referenciar a informações provenientes de outras camadas. Estas

informações de outras camadas podem ser de textura, vetores de movimento ou resíduo, o que torna este bloco mais complexo.

Os resultados do módulo de predição podem passar por uma etapa de “Refinamento de ½ pixel” (*half pixel*) e/ou “Refinamento de ¼ pixel” (*quarter pixel*). A decisão por usar ou não os módulos de refinamento de pixel é tomada por um módulo de “Decisão”, que avalia as distintas opções para determinar a mais adequada (menor erro residual). Os vetores escolhidos já podem ser enviados para compor o fluxo de saída comprimido, ao mesmo tempo em que passam pelo algoritmo de “Compensação”, que é responsável por reconstruir a imagem a partir destes vetores e com isso possibilitar o cálculo das informações de resíduo.

Tanto as informações de resíduo como as de textura de quadros intra devem passar pelo algoritmo de “Decomposição Temporal” para gerenciar e sincronizar as mesmas. A seguir estas informações passam pelos algoritmos de compressão espacial (em sequência, os módulos DCT, Hadamard e Quantização), gerando assim uma camada comprimida.

As informações geradas por estes módulos são enviadas para o módulo de entropia (CABAC ou CAVLC), a fim de que sejam comprimidas visando gerar o fluxo de saída. Além de serem enviadas para o módulo de entropia estas informações devem ser também usadas pelos módulos computacionais inversos (IDCT, IHAD e IQUANT), que têm por função reconstruir a imagem codificada, após passar pelo “Filtro Anti-Blocagem” e com isso determinar as novas imagens de referência.

Para produzir as diversas camadas de qualidade cada módulo deve implementar a codificação complementar (diferença entre o vídeo da camada e o reconstruído pela camada inferior) considerando-se a relação dos distintos valores de QP.

No codificador escalável a imagem reconstruída não se limita apenas ao cálculo de resíduos, uma vez que possivelmente será também usada como referência para a codificação realizada sobre a imagem da camada superior.

O fluxo de saída comprimido é lido pelo módulo de “Controle de Taxa”, para ajuste dinâmico da banda ocupada variando o parâmetro de quantização (QP) de cada camada. Por fim, os dados gerados pela codificação de cada uma das camadas (informações da camada base e das de refinamento) são enviados para um módulo “Multiplexador”, o qual será responsável por identificar e empacotar as informações das diferentes camadas em um fluxo único de saída. Maiores detalhes sobre o funcionamento internos destes módulos são apresentados no capítulo a seguir (Seção 4.2.2).

4 DESENVOLVIMENTO METODOLÓGICO DA PROPOSTA

4.1 BREVE ANÁLISE DO CENÁRIO TECNOLÓGICO

O contínuo crescimento do número de transistores por circuito integrado oferece cada vez mais recursos para projetos de sistemas digitais complexos. Neste contexto, são abertos grandes campos de pesquisa na área de VLSI, como por exemplo:

a) redução do custo de fabricação, principalmente para baixo volume de produção;
b) desenvolvimento de metodologias de projeto que reduzam o tempo de projeto;
c) estudo de arquiteturas que melhor utilizem os recursos disponíveis no circuito.

Dentre as tecnologias mais difundidas merecem especial destaque os microprocessadores de propósito geral, que evoluíram de forma sistemática desde a década de 70. Mesmo assim o que se percebe é que apesar da contínua evolução das tecnologias não se tem conseguido reverter estes recursos em ganhos proporcionais de desempenho, ou seja, o ganho do desempenho das unidades de processamento central, não acompanha, na mesma proporção, seu aumento de densidade (CHEN et al., 2009). Para ilustrar isso com um exemplo, a Figura 34 apresenta, de forma gráfica, a evolução histórica das tecnologias Intel x86 comparando o número de transistores em relação ao desempenho geral obtido.

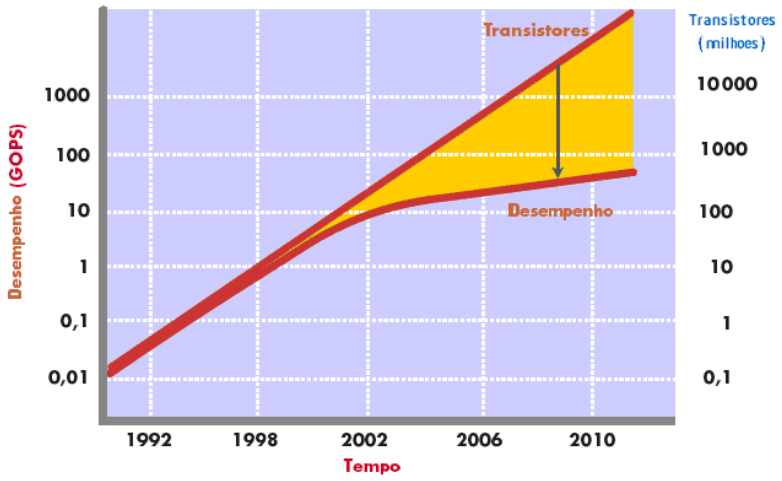


Figura 34: Evolução das CPUs comparando área ocupada pelo desempenho

Essa limitação de desempenho observada torna, muitas vezes, difícil a implementação em tempo real de modernos codificadores de vídeo através de soluções inteiramente por software, devido à elevada carga computacional exigida por estes. Lin (2006) aponta que um codificador H.264/AVC para trabalhar com vídeos de alta definição (HDTV) requer até 3,6 tera instruções por segundo (TIPS) processando 5,6 terabytes de dados por segundo.

Se as demandas já são elevadas para codificadores de vídeo convencionais, para um codificador escalável como o H.264/SVC, que inclui novos algoritmos (reescalonamento de vídeos, módulo de pré-processamento MCTF, procedimento de predição aprimorado, entre outros) e traz a necessidade de operar, no mesmo intervalo de tempo, com múltiplas camadas de vídeo, estas demandas se mostram ainda maiores. Com isso, torna-se basicamente impraticável sua implementação em uma estratégia unicamente por software e centralizada em um processador de propósito geral (BERTOZZI et al., 2005).

Aplicações como esta, que exigem alto custo computacional e o tratamento de grandes volumes de dados, tendem a ser implementadas de forma mais eficiente quando se conta com o apoio de arquiteturas de aceleração em hardware, tais como componentes dedicados (ASICs – *Application Specific Integrated Circuits*), tecnologias de múltiplos núcleos como é o caso de plataformas recentes de processamento digital de sinais (DSP – *Digital Signal Processing*) ou unidades de processamento gráfico (GPU – *Graphical Processing Unit*) ou arquiteturas de lógica programável (CHEN et al., 2009).

Com base neste conceito, nas últimas décadas, surgiram diversas iniciativas que propõem o trabalho colaborativo entre hardware e software, onde componentes de hardware dedicados são utilizados para apoio a microprocessadores convencionais (WOLF, 2003).

Em uma estratégia de projeto colaborativo, os elementos de hardware normalmente se encarregam de implementar os algoritmos computacionalmente mais complexos, enquanto que o software, que roda em um microprocessador de propósito geral (CPU – *Central Processing Unit*), se encarrega das funções de alto nível, tais como gerência operacional, controle de sistema de arquivos e interação com o usuário. Assim, essa abordagem consegue, na prática, aproveitar o melhor de cada tecnologia: elevada eficiência computacional das plataformas de hardware aliada à redução no tempo de desenvolvimento e interoperabilidade, que são características dos softwares (DE MICHELI; EMST; WOLF, 2002).

Para se obter, entretanto, resultados mais eficientes ao se adotar uma solução desse tipo é necessário um conhecimento aprofundado da aplicação alvo, a fim de se identificar as partes que, de fato, devem ser implementadas em hardware ou software. Se esta escolha não for feita

de forma adequada, o resultado final pode ser o desenvolvimento de plataformas mais complexas e que ainda não atendam aos requisitos esperados (JERRAYA; WOLF, 2005).

Historicamente, as primeiras soluções deste tipo se caracterizavam pelo uso placas dedicadas, que incluíam componentes como CPUs e ASICs, especificamente escolhidos para atender a uma dada aplicação alvo. Na prática, a criação de um sistema de co-projeto HW-SW desse tipo acaba se tornando complexa, mesmo para aplicações tradicionais de mono-processamento (*single core*), pois seu projeto envolve plataformas que consistem de diversos componentes (processador, memórias e ASIC's). O projetista, durante o desenvolvimento da solução, deve observar diversas questões práticas, como a taxa de dados suportada, níveis de tensão e consumo de corrente individual, compartilhamento de memória, freqüências de operação, interfaces de comunicação, entre outras (NOTEBAERT; COCK, 2004). Essas soluções, normalmente, conseguem gerar um significativo aumento de desempenho quando comparadas com soluções convencionais (somente CPU), uma vez que ASICs são projetados para desempenhar aplicações específicas com elevada eficiência. Por outro lado, representam sistemas de baixa flexibilidade, ou seja, são voltados para atender a uma determinada finalidade e sempre que forem exigidas mudanças na aplicação (como, por exemplo, inclusão de novas versões de algoritmos), exige-se uma reformulação das placas de hardware, substituindo-se ligações e mesmo componentes eletrônicos (ATITALLAK et al., 2011).

Soluções mais genéricas propõem o uso de componentes de DSP no lugar de ASICs. Componentes de DSP nem sempre conseguem atingir o mesmo desempenho de um ASIC dedicado, porém trazem a vantagem de que uma mesma placa pode ser utilizada para diversas aplicações (bastando trocar o programa que é executado nos núcleos de DSP). A estrutura de programação desses módulos pode se basear em técnicas de VLIW (*Very-Large Instruction Word*), quando os compiladores promovem diferentes técnicas para otimizar o desempenho, tais como reordenamento de instruções, planejamento antecipado de desvios e aproveitamento inteligente de registradores e recursos de memória, e assim como o uso de operações do tipo SIMD (*Single Instruction Multiple Data*), que permitem a operação simultânea sobre vários dados em um mesmo ciclo de relógio (CHEN et al., 2009). Por apresentarem arquiteturas especialmente dedicadas para implementações de algoritmos iterativos e vetoriais estas soluções comumente tendem a apresentar um significativo ganho de desempenho quando comparado com arquiteturas microprocessadas de propósito geral. Entretanto, para se obter o máximo desempenho das plataformas baseadas em DSP, é importante que os algoritmos sejam adequadamente adaptados para explorar as características destes componentes (larguras

de registradores, tipos de memórias internas, instruções de SIMD disponíveis, etc), o que, muitas vezes, leva a um aumento do tempo nas etapas de programação, depuração e aprimoramento de algoritmos (NARVEKAR et al., 2009).

Outra alternativa que tem sido proposta recentemente para explorar o paralelismo em nível de software é a GPGPU (*General Purpose Graphical Processing Unit*), que propõe o uso de GPUs para processamento não-gráfico. GPUs são componentes criados originalmente para o mercado de placas de aceleração gráfica, incorporando, em uma mesma arquitetura, diversos multiprocessadores, cada um deles contendo múltiplos núcleos de execução paralela. Se o algoritmo implementado possuir grande simetria sobre vetores de dados e operações, a arquitetura de GPU pode executar centenas de tarefas simultâneas. Seus ganhos entretanto estão fortemente baseados na forma como os algoritmos são organizados para prover operações multi-tarefas (*multi-threads*), o que leva muitas vezes à reescrita dos algoritmos. Além disso, a comunicação entre as múltiplas GPUs ou entre GPUs e memórias se torna um ponto chave. Fontes de atraso, tais como latências de comunicação física, dificuldade de sincronização e conflitos no acesso às memórias podem restringir ou, até mesmo, anular os ganhos de desempenho obtidos pelas múltiplas tarefas (BILGIC et al, 2010).

Outra tecnologia muito relevante para o mercado de alto desempenho são as lógicas programáveis, que permitem o desenvolvimento de algoritmos específicos no contexto dos circuitos digitais. A solução se mostra muito flexível, pois a adaptação da plataforma ocorre em nível de hardware, sem passar pela reformulação das ligações externas de componentes nas placas, mas sim pela reprogramação das ligações internas entre blocos lógicos. Do ponto de vista funcional, as ligações internas dessas lógicas são definidas por complexas redes de transistores, que podem ser ativados por fusíveis (tecnologia CPLD), quando uma única programação é possível, ou por bits de memória (tecnologia FPGA), que permite a reprogramação do componente(SAYED; BADAWI; JULLIE, 2008).

A reconfiguração pode ser global (quando todas as interconexões internas são rearranjadas) ou parcial (apenas parte das interconexões internas é alterada). Essa flexibilidade se tornou uma grande vantagem da tecnologia, pois permite que a mesma plataforma possa ser usada para múltiplas aplicações atingindo-se desempenhos tão elevados quanto os obtidos com ASICs (ATITALLAH et al., 2011).

4.2 METODOLOGIA DE PROJETO

Baseado nas alternativas estudadas, foi desenvolvido um projeto de codificador de vídeo escalável, baseado diretamente na tecnologia de lógicas programáveis. De forma geral, a solução desenvolvida tem a parte de software implementada sobre uma plataforma de computador pessoal, onde roda o software de referência do padrão H.264/SVC, enquanto que a parte de hardware do sistema é caracterizada por uma placa de desenvolvimento interconectada a este, a partir de uma interface de comunicação de alta velocidade.

Projetos colaborativos de hardware e software como este são relativamente complexos e fortemente dependentes da aplicação alvo. A eficiência de um projeto desses depende das demandas exigidas pela aplicação (desempenho, custo de produção, facilidade de programação, entre outros) e da forma como o projeto faz o uso dos componentes de hardware e software (DE MICHELI; EMST; WOLF, 2002).

Devido a esta complexidade inerente, a metodologia de projeto adotada englobou diferentes etapas complementares, buscando-se isolar as diversas questões envolvidas com este projeto e assim facilitar as tomadas de decisão. De forma resumida a metodologia de projeto foi dividida nas seguintes etapas:

a) Modelagem;
b) Particionamento;
c) Integração;
d) Avaliação.

A etapa de modelagem consiste na especificação e refinamento dos algoritmos internos do sistema a ser desenvolvido. Durante esta etapa foi realizada uma análise detalhada da implementação de um codificador de vídeo escalável padrão H.264/SVC, para fins de especificação das reais demandas computacionais deste. Diversos experimentos práticos foram realizados visando identificar o funcionamento e as configurações dos principais algoritmos internos do codificador. Ao final desta etapa pode-se gerar um modelo refinado do codificador, ajustado para garantir maior desempenho sem perdas significativas de qualidade, o qual foi assim tomado como base para a implementação desta solução (aplicação alvo).

Já a etapa de particionamento, ou mapeamento HW/SW, é responsável por tomar a aplicação alvo especificada na etapa anterior e definir as partes internas desta que serão desenvolvidas em software e as que devem ser implementadas em hardware.

Para a realização desta etapa, o codificador H.264/SVC foi dividido em diferentes blocos funcionais, que foram a seguir avaliados, procurando-se identificar questões práticas como dependências de dados e potenciais de paralelismo. Com base nestas análises pode-se definir, de forma mais clara, quais módulos seriam mantidos em software e quais seriam portados para hardware.

A etapa de integração envolve a definição das interfaces, estruturas de dados e estratégia de sincronização adotadas para as trocas de informação entre as entidades de software e hardware. Inicialmente, foi feita uma análise teórica das principais estratégias de comunicação e sincronismo entre entidades de hardware e software, considerando as questões próprias da solução proposta (tecnologias envolvidas), o que serviu para indicar as alternativas mais adequadas. A definição final a ser adotada, porém, dependia fortemente da plataforma a ser adotada e, por isso, só foi fechada após a realização de experimentos práticos em laboratório.

Por fim, foi realizada uma etapa de avaliação teórica e funcional da solução. A avaliação teórica define, de forma prática, se a solução final conseguiu atender aos requisitos da aplicação alvo, considerando-se especificações da norma H.264/SVC. Esta avaliação é importante também para guiar a escolha da plataforma de trabalho.

Posteriormente foi realizada a avaliação funcional da aplicação, que somente pode ocorrer após se passar pela realização física (integração) das diversas entidades envolvidas, ou seja concluídas as atividades de síntese da parte de hardware e a compilação da parte de software (CHEN et al, 2009).

Como forma de ilustrar graficamente a metodologia de projeto adotada, a Figura 35 traz uma representação das diferentes etapas envolvidas desta solução de co-projeto. Na figura a parte de hardware é definida por módulos desenvolvidos usando linguagem de descrição de hardware VHDL (VHSIC *Hardware Description Language*) enquanto que os módulos de software foram implementados em C++.

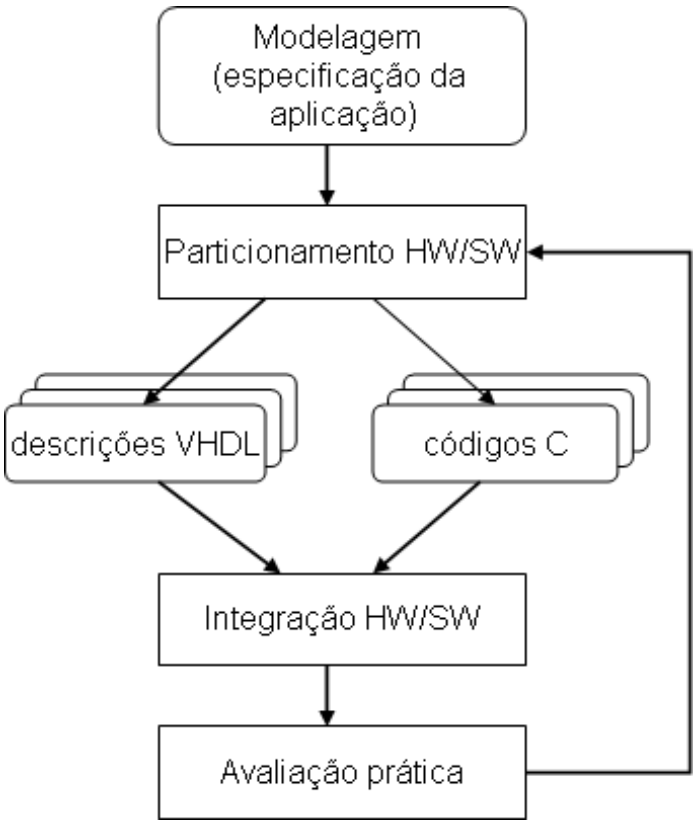


Figura 35: Etapas genéricas de um algoritmo de co-projeto de HW-SW

Conforme demonstrado na Figura 35, durante o período do projeto as etapas de particionamento, integração e avaliação, de acordo com os resultados finais obtidos passam por etapas de refinamento, buscando-se aprimorar as partes de software e hardware, de forma a otimizar a arquitetura de cada módulo em função de seu algoritmo específico. No caso todos os refinamentos propostos visaram privilegiar o aumento de desempenho e redução dos efeitos causados por atrasos associados com a interface entre computador e a plataforma de hardware.

Dados comparativos de diferentes alternativas testadas, bem como a avaliação final de seus resultados funcionais, são demonstrados na seção de experimentos deste documento (Capítulo 6).

A seguir se apresenta, de forma discriminada, o detalhamento de cada uma destas etapas envolvidas com a metodologia do projeto em questão.

4.2.1 Modelagem da Aplicação Alvo

A primeira etapa a ser desenvolvida por qualquer projeto colaborativo de hardware e software deve ser a especificação das diversas funcionalidades internas da aplicação alvo. Estas especificações são importantes, pois determinam os algoritmos a serem implementados e suas principais demandas, os quais posteriormente serão distribuídos entre blocos de software ou hardware (JERRAYA; WOLF, 2005).

Como já demonstrado, uma aplicação de codificação de vídeo escalável é bastante complexa. A fim de mensurar na prática as demandas de um codificador escalável padrão H.264/SVC, e possivelmente refinar o modelo arquitetural desta solução, foram realizados diversos experimentos.

Durante a elaboração destes experimentos procurou-se avaliar, para diferentes configurações, algumas das principais características práticas importantes para codificadores, tais como qualidade do vídeo gerado, complexidade de implementação e desempenho (tempo total de codificação).

Para a realização destes experimentos práticos foi utilizado o software JSVM versão 9.19 (JOINT SCALABLE TEAM MODEL, 2010). Esse software foi desenvolvido pelo grupo JVT para servir de referência para desenvolvedores e usuários do padrão SVC.

Todos os ensaios realizados foram repetidos com diferentes seqüências de vídeo, especialmente escolhidas por apresentarem distintos níveis de detalhamento e movimentação, visando assim aumentar a consistência dos dados obtidos. Particularmente, as seqüências escolhidas foram BUS, CITY, FOOTBALL, FOREMAN e MOBILE, que apresentam distintos padrões de movimentação, a fim de se ter uma avaliação mais ponderada dos algoritmos frente a diferentes situações de entrada.

Para os experimentos realizados, todas as sequências de entrada tinham como características: uso do formato YUV 4:2:0, com resolução espacial CIF (352x288 pixels) e taxa de exibição de 15 fps, os quais foram codificados para uma taxa de bits de saída constante de 1Mbit/s.

A escolha de se trabalhar com baixas resoluções (CIF) e taxas de apresentação (15 fps) visou reduzir o tempo dos ensaios, bem como facilitar a comparação com outros trabalhos científicos, que usam os mesmos vídeos.

Para simplificar a apresentação dos resultados, nesta seção são apresentados apenas os gráficos que resumem os resultados obtidos. As informações quantitativas completas obtidas com estes ensaios (tabelas que discriminam resultados para diferentes vídeos e componentes de pixel) se encontram apresentadas no APÊNDICE A.

4.2.1.1 Experimento sobre Escalabilidade Temporal

O primeiro ensaio realizado buscou avaliar as características próprias do método de escalabilidade temporal do padrão H.264/SVC. A codificação escalável temporal neste padrão é obtida a partir de um mecanismo inovador de decomposição temporal, o qual foi incluído na estrutura do codificador visando aumentar a eficiência da codificação. Por estar incluso diretamente na estrutura do codificador SVC, o mecanismo de escalabilidade temporal acaba se tornando parte inerente do codificador, ou seja, sua operação não pode ser desabilitada ou removida. O mesmo não acontece com os demais tipos de escalabilidade (espacial e SNR), que podem ser habilitados ou desabilitados (RIECKL, 2008).

O operador pode, entretanto, alterar as configurações de funcionamento desta codificação escalável temporal, através da alteração do tamanho do GOP.

Devido ao mecanismo B hierárquico utilizado pelo codificador, o ajuste deste parâmetro tem efeito direto sobre o número de camadas geradas. Por exemplo, um vídeo com valor de GOP igual a 16 produz cinco camadas temporais (T0, T1, T2, T3 e T4). Já um vídeo com GOP igual a 8 leva à geração de apenas quatro camadas temporais (T0, T1, T2 e T3). O efeito colateral de se reduzir o número do GOP é que se reduz também a eficiência da codificação. Isso porque, com a redução do número de quadros que compõem cada GOP, aumenta-se proporcionalmente a incidência de quadros I. Os quadros I apresentam, em média, uma menor taxa de compressão, visto que não exploram a redundância temporal entre quadros.

Os resultados obtidos com este primeiro ensaio são apresentados na Tabela 29 (APÊNDICE A) onde se apresentam discriminados os dados de tempo de execução e PSNR para cada um dos componentes Y, U e V do vídeo final reconstruído.

Avaliando-se os resultados obtidos percebe-se uma variação significativa entre os resultados das diferentes seqüências de vídeo, porém com uma relação bastante similar entre as configurações utilizando GOP igual a 8 ou GOP igual a 16. Para facilitar a visualização desta relação, a Figura 36 apresenta, de forma gráfica, os resultados de PSNR obtidos com este primeiro ensaio para a componente Y (luminância).

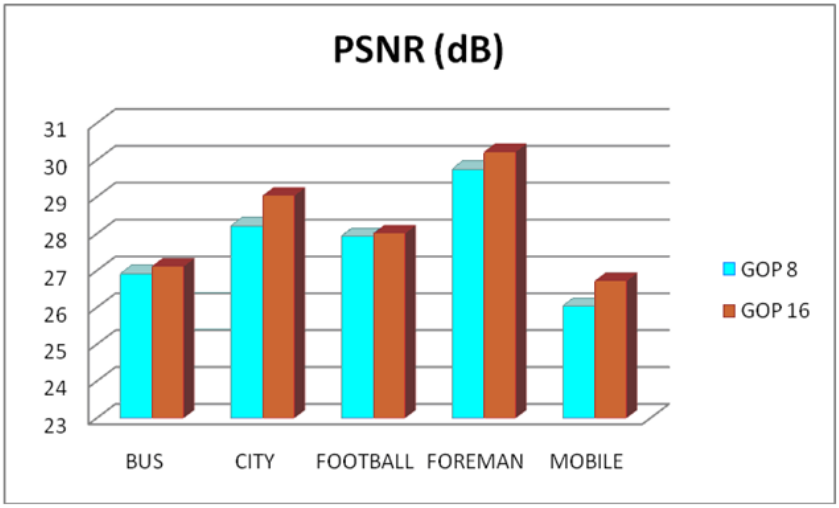


Figura 36: Comparação da qualidade de codificações escaláveis temporais

De forma geral a eficiência de codificação pode ser considerada baixa para a escalabilidade do tipo temporal (valores médios abaixo de 29 dB). Quando se busca, porém, a implementação prática de um codificador, além da qualidade final, deve-se avaliar outros parâmetros tais como complexidade computacional (recursos de memória e operadores utilizados) e/ou desempenho do algoritmo. Neste sentido, procurou-se, durante estes ensaios, registrar dados adicionais que complementassem a informação de qualidade. A análise de desempenho de um algoritmo, por exemplo, pode ser estimada a partir do tempo de execução do mesmo. Para tanto cada uma das configurações de codificação (GOP igual a 8 e GOP igual a 16) foi executada sobre a mesma máquina, permitindo-se comparar a diferença do tempo de execução. A máquina utilizada possui como configurações: processador Pentium D 3,4GHz, com memória cache L1 de 16kB, memória cache L2 de 2MB, memória DDR2 FSB 666MHz de 2GB e HD SATA 7200 RPM de 160GB. Obviamente os valores dos tempos devem variar ao se utilizar diferentes topologias de hardware (velocidade da CPU, memórias DRAM, memórias cache, etc). Entretanto, estima-se que, independente dos valores individuais, a relação registrada entre as duas configurações deve ser basicamente a mesma.

Os resultados de tempo de execução medidos são apresentados na Figura 37.

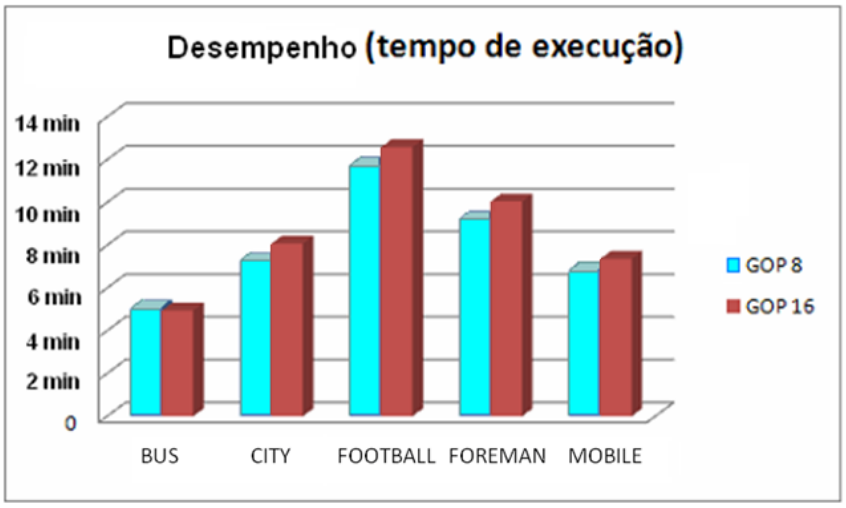


Figura 37: Comparação de desempenho de codificações escaláveis temporais

Conforme esperado, a utilização de um GOP igual a 8 em detrimento de um GOP de 16 causa uma pequena degradação de qualidade, devido à menor eficiência de compressão. O desempenho do codificador, entretanto, melhora ao se trabalhar com GOPs menores, uma vez que se reduz o número de operações de predição. A redução do tempo de execução chega a 15%, em alguns casos, o que pode ser vantajoso para aplicações críticas no tempo.

4.2.1.2 Experimento sobre Escalabilidade Espacial

O segundo ensaio realizado visou determinar as demandas da escalabilidade espacial do padrão SVC, ou mais especificamente da especificação ESS, que foi elaborada pelo padrão visando prover maior flexibilidade a aplicações de escalabilidade espacial (suporte a diferentes posicionamentos e relações de resolução entre camadas).

Na prática, para a realização deste experimento foram gerados três cenários de fluxos escaláveis, dois incluindo o algoritmo ESS e outro sem utilizá-lo. O objetivo foi analisar o desempenho do algoritmo ESS em duas situações distintas: caso diádico e caso não diádico.

No caso diádico o vídeo original tem resolução de 352x288 pixels e o vídeo subamostrado 176x144 pixels, enquanto que o caso não diádico o vídeo subamostrado tinha resolução de 240x130pixels.

A partir da análise destes resultados pode-se avaliar a real influência da extensão espacial sobre um codificador SVC. Conforme comentado, os ensaios realizados procuraram

analisar questões práticas relevantes, tais como qualidade do vídeo reconstruído, tempo de codificação e complexidade dos seus algoritmos internos.

Utilizou-se como referência para estas avaliações os vídeos reconstruídos pelo processamento da primeira camada de enriquecimento (camada 1), que para todos os cenários adotavam a mesma resolução de saída (formato CIF), tornando assim as comparações entre métodos válidas. Os resultados obtidos com este ensaio são listados na Tabela 30 (APÊNDICE A) onde se encontram discriminados os dados de tempo de execução e PSNR para cada um dos elementos Y, U e V. Os resultados da tabela trazem os dados de tempo de execução medidos nos ensaios em duas colunas de tempo. Foi discriminado assim, pois o software JSVM não implementa o mecanismo de subamostragem dentro do codificador SVC, sendo necessário utilizar-se um conversor externo, que é fornecido como um software adicional ao software de referência JSVM (JOINT SCALABLE TEAM MODEL, 2007).

A avaliação dos resultados obtidos aponta para variações muito pequenas do parâmetro PSNR, sendo ligeiramente superior (cerca de 0,2dB) quando se utiliza o algoritmo ESS (diferença maior registrada apenas para a seqüência CITY).

Uma conferência visual destes resultados comentados pode ser observada na Figura 38.

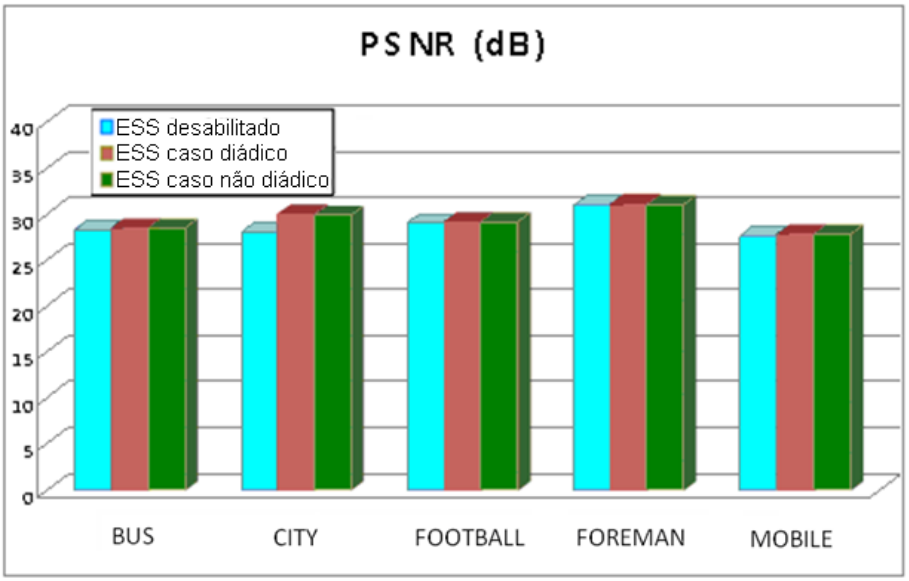


Figura 38: Comparação da qualidade da camada 1 em relação à extensão ESS

Para verificação de desempenho do codificador para os três cenários de escalabilidade espacial, o tempo total de execução foi calculado como a soma dos tempos de execução

medidos para ambos os algoritmos (codificador e conversor externo). A Figura 39 traz a apresentação dos resultados obtidos na forma de gráficos de barra. Percebe-se que o tempo de execução da configuração sem ESS é significativamente menor.

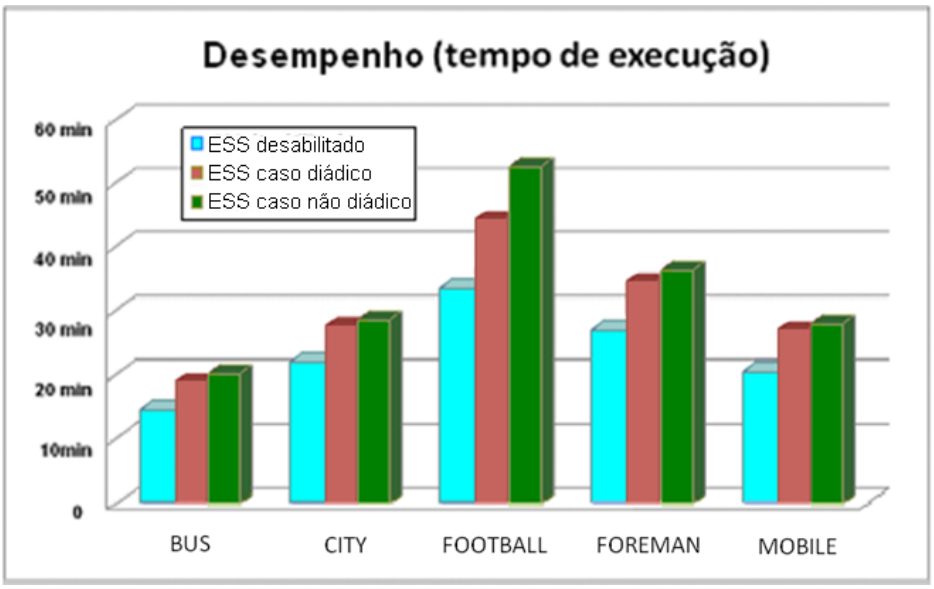


Figura 39: Comparação de desempenho referente à extensão ESS

A última análise feita diz respeito à avaliação das demandas computacionais dos algoritmos. Na prática, a avaliação precisa das demandas por recursos computacionais de um dado algoritmo não é uma tarefa simples, pois depende de inúmeros fatores, muitos dos quais são específicos de cada implementação, tais como recursos da arquitetura de hardware (largura de registradores, módulos internos e barramentos), metodologia e estilo de projeto, nível de paralelismo adotado, entre outros. Pode-se, entretanto, obter uma idéia aproximada desta demanda avaliando-se o uso de operadores lógicos, somadores, multiplicadores e memória. Esta metodologia foi proposta pelo grupo JVT (2002) para avaliar a complexidade de implementações H.264/AVC, onde se considera que a complexidade de algoritmos descritos em linguagem C pode ser estimada por quatro grupos de operadores:

- *Lógicos*: correspondente aos operadores “!”, “&&”e “||”;
- *Comparação*: referente aos operadores “==”, “!=”, “>”, “>=”, “<” e “<=”;
- *Memória*: corresponde aos operadores “=”, “[]”, “->” bem como ponteiro “*”;
- *Aritméticos*: demais operadores (“+”, “-”, “*” e “/”).

A fim de extrair estas informações do software JSVM, foi inicialmente utilizada a ferramenta GPROF, que consegue mensurar em tempo de execução das funções utilizadas por

um dado software (ATITALLAH et al., 2011). A partir da discriminação dessas funções foi desenvolvido um software próprio para realizar a leitura e localização automática de operadores dentro dos códigos fontes das funções indicadas, listando o número total de ocorrências. A elaboração desse software precisou levar em consideração o contexto onde se encontram os diferentes operadores localizados, a fim de evitar uma interpretação falsa dos mesmos. Por exemplo, os operadores de multiplicação e divisão (“*” e “/”), muitas vezes, são utilizados em comentários (como “/*”, “//” ou “*/), devendo ser, nestes casos, descartados da avaliação de complexidade. O resultado do software é uma saída em um arquivo texto como:

```
Operadores no cenario A (com ESS desativado)
 +-   *   /  <<>>  &&  ||   !  =  *p  ->   []   ==   !=  <=>
7451 828 204  714 2227 285  0  0  16 3590 6415 1076  396 1196

Operadores no cenario B (ESS caso diadico)
 +-   *   /  <<>>  &&  ||   !  =  *p  ->   []   ==   !=  <=>
8067 942 238  892 2295 303  0  0  16 3776 6582 1113  406 1305

Operadores no cenario C (ESS caso nao diadico)
 +-   *   /  <<>>  &&  ||   !  =  *p  ->   []   ==   !=  <=>
8099 948 238  896 2295 304  0   0  16 3786 6588 1115  406 1312

Operadores no conversor de resolucao (somar ao ESS)
 +-   *   /  <<>>  &&  ||   !  =  *p  ->   []   ==   !=  <=>
236   71  25  115  33   4   0  0   0   45  63   19    3   42
```

Fazendo-se os devidos agrupamentos dos operadores listados chega-se ao resultado indicado na Figura 40.

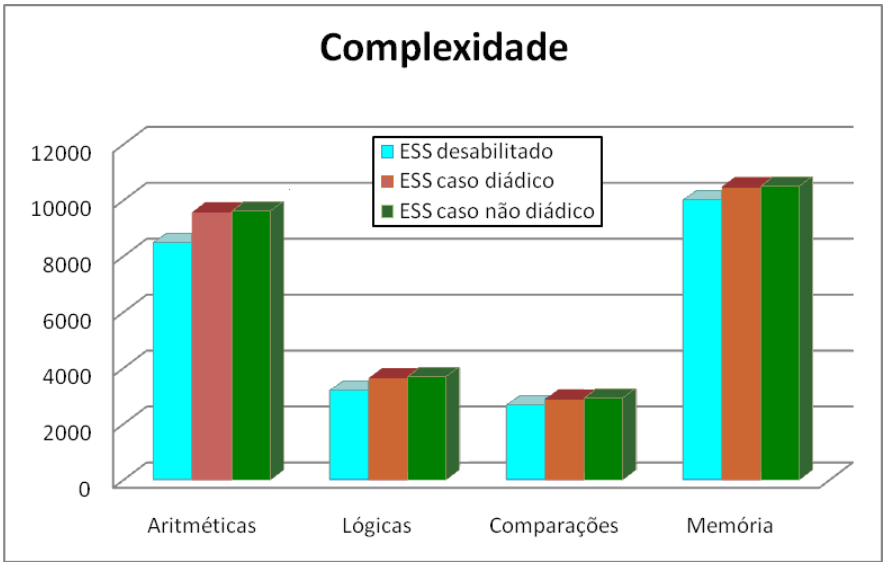


Figura 40: Avaliação da complexidade computacional da extensão ESS

A partir destes dados, pode-se visualizar o impacto computacional de um codificador SVC ao se incluir os algoritmos de codificação escalável espacial.

Comparando-se os resultados obtidos constata-se que o uso da escalabilidade espacial gera aumentos significativos no consumo de operadores (cerca de 17% para o caso de operações aritméticas) e tempo de execução (cerca de 47% no pior caso), sem ganhos significativos de qualidade (menor que 0,2dB em média).

Estes resultados apontam para que a utilização deste recurso deve ser reservada para situações onde a adoção de camadas com diferentes resoluções espaciais seja, de fato, necessária.

4.2.1.3 Experimento sobre Escalabilidade SNR

Os próximos ensaios realizados visaram identificar as características específicas das diferentes metodologias de implementação de uma codificação escalável do tipo SNR.

Na prática, são três as metodologias possíveis para a escalabilidade SNR: CGS, MGS e FGS.

Na prática, porém, a metodologia FGS não chega a ser uma abordagem concorrente às duas demais, pois visa uma aplicação diferenciada (adequação quase contínua do fluxo codificador em relação a variações dinâmicas do meio de comunicação).

Considerando essa questão e também visando-se realizar uma comparação mais justa, os ensaios de escalabilidade SNR realizados consideraram apenas as metodologias CGS e MGS.

Os resultados obtidos com os ensaios práticos da escalabilidade SNR são apresentados na Tabela 31 (APÊNDICE A):

A avaliação da qualidade obtida pelas duas metodologias indicou que a versão MGS gera um vídeo de melhor qualidade para ambas as camadas, porém com diferenças mais significativas para a camada mais baixa (camada base).

Isso pode ser observado nos gráficos a seguir (Figura 41 e 42) que discriminam os valores de PSNR referentes ao componente Y reconstruído da camada 0 (base) e 1 (primeira camada de enriquecimento), respectivamente.

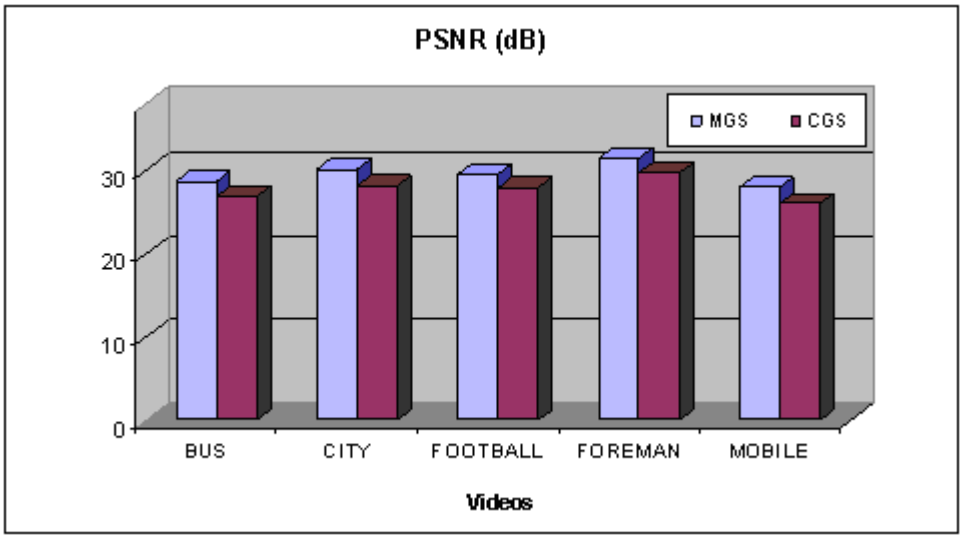


Figura 41: Comparação da qualidade da camada base para escalabilidade SNR

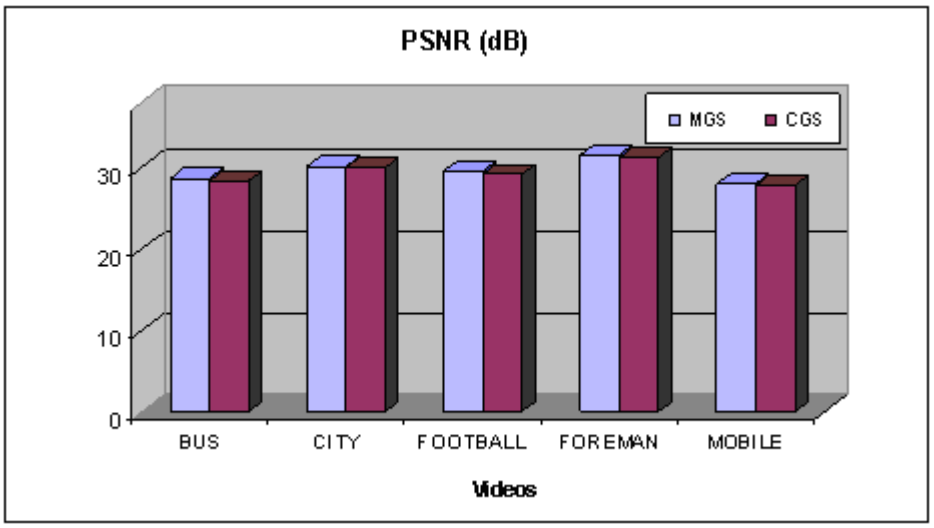


Figura 42: Comparação da qualidade da camada 1 para escalabilidade SNR

A avaliação de desempenho (tempo total de codificação) entre as duas metodologias indica que a versão MGS é, de fato, mais rápida, o que pode ser visualmente identificado na Figura 43.

Assim sendo, a metodologia MGS se mostra a mais indicada para uma implementação prática (melhor qualidade com menor tempo de execução).

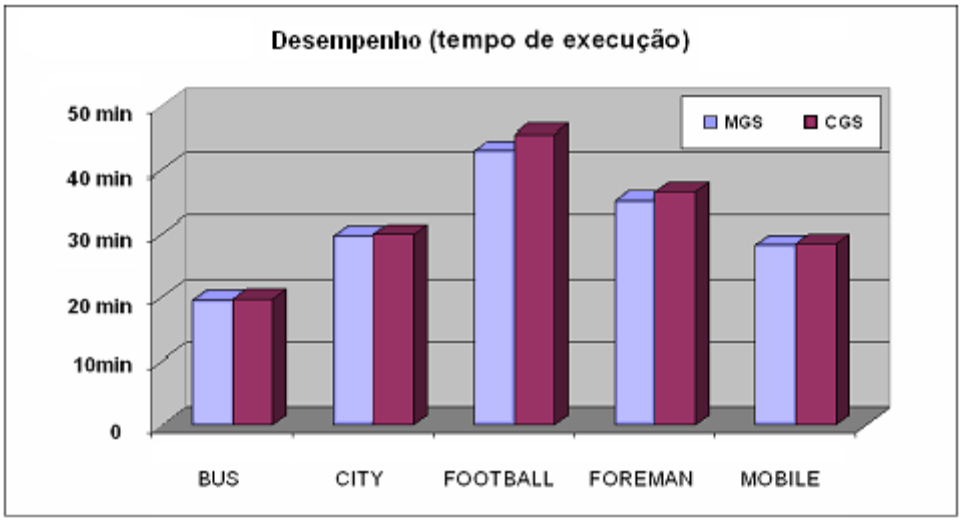


Figura 43: Comparação de desempenho referente à escalabilidade SNR

A análise da demanda computacional (número de operadores) das duas metodologias não registrou mudanças significativas. Isso se explica, pois a principal diferença entre as duas estratégias reside basicamente na forma como se implementa internamente o mecanismo de predição entre quadros, sem que para isso seja inserido nenhum módulo adicional.

4.2.1.4 Experimento sobre Mecanismos de Predição

Os novos ensaios realizados procuraram avaliar diferentes estratégias de predição entre quadros a fim de verificar na prática as influências que estas causam sobre um codificador. Inicialmente procurou-se fazer ensaios relacionados com as regiões usadas como referência para a predição.

A configuração padrão do JSVM, chamado nos experimentos de cenário normal, utiliza mecanismo de busca em diamante (PORTO, 2008) com janela de busca de 96x96 pixels e bipredição habilitada (ou seja, se referencia a quadros passados e futuros).

A seguir foram feitos outros ensaios mudando a estratégia de deslocamento sobre a imagem de referência, reduzindo a quantidade de quadros de referência e variando a janela de busca.

O resultado destes ensaios aparece representado na Tabela 32 (APÊNDICE A).

Em outra configuração se avaliou a substituição do mecanismo de pesquisa em diamante pelo método de busca completa (RICHARDSON, 2003). A terceira configuração não trabalha com quadros passados e futuros (bipredição), mas apenas quadros do passado. A quarta configuração adota uma janela de busca de 24x24, enquanto que a última configuração avaliada expande a janela de busca para uma área de 192x192 pixels.

Apesar das diferenças claras entre cenários, de forma geral pode-se perceber que todos cenários apresentam resultados de PSNR bem próximos (cerca de 0,05dB), o que pode ser visualizado na Figura 44.

A representação gráfica escolhida foi da seqüência FOREMAN, porém outras seqüências trazem resultados similares.

VIDEO FOREMAN

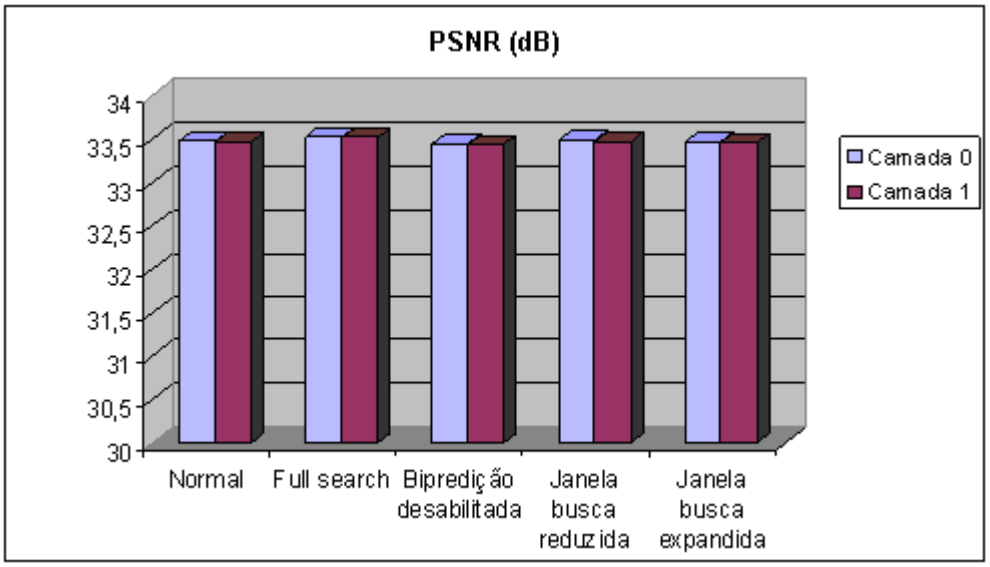


Figura 44: Comparação da qualidade para diferentes configurações de predição

Apesar de similares em relação à qualidade do vídeo produzida, a variação registrada pelos diferentes cenários, em termos do tempo de execução, é bastante grande (conforme apresentado na Figura 45).

Nesta figura, foram omitidos os dados da configuração de busca completa, visto que este cenário apresenta um tempo de execução dezenas de vezes superior aos demais, o que representaria uma comparação desproporcional.

Dentre os cenários restantes apresentados percebe-se que a topologia com janela de busca reduzida é a que apresenta menor tempo de execução.

VIDEO FOREMAN

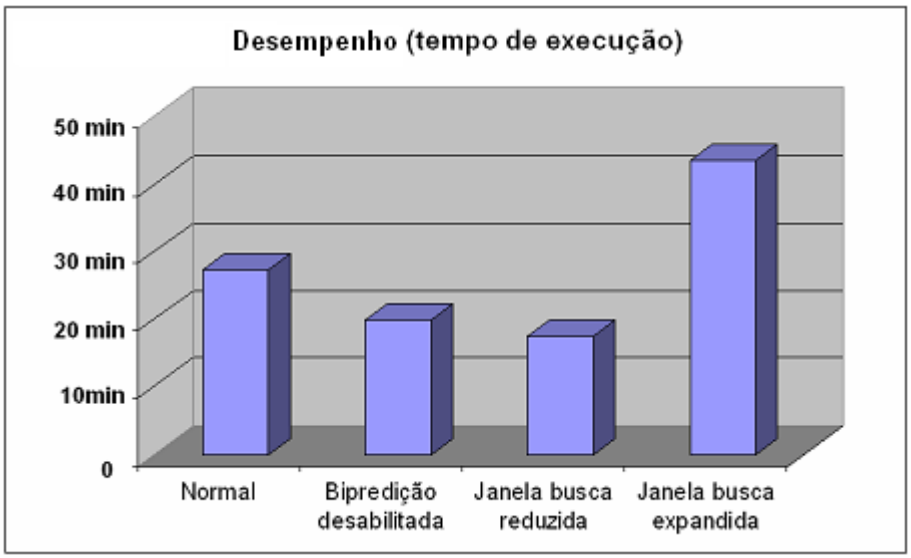


Figura 45: Comparação de desempenho para diferentes configurações de predição

A análise realizada aponta para a redução da janela de busca como um recurso que causa aumento significativo do desempenho do codificador (redução do tempo de codificador) sem perda significativa de qualidade. A desabilitação do recurso de bipredição também aumenta o desempenho do codificador em mais de 70% com perdas sutis de qualidade (menores que 0,2dB).

Os próximos ensaios realizados visam identificar as melhores configurações de particionamento de dados e refinamento de pixels, recursos que tendem a aumentar a eficiência de codificação quando habilitados, porém causando aumentos significativos do tempo de codificação. Estes resultados foram importantes para identificar a topologia mais adequada para uma implementação de alto desempenho (menor tempo de codificação), sem resultar em grandes perdas de qualidade.

Os resultados quantitativos para cada um dos diferentes cenários analisados são apresentados na Tabela 33 (APÊNDICE A), discriminados por legenda. Os resultados obtidos não foram apresentados na forma de gráficos dada a grande quantidade de cenários analisados.

De forma geral, os resultados obtidos nos ensaios realizados demonstraram que a utilização de blocos de 8x8 pixels e 16x16 pixels como elementos de partição, levam a um aumento significativo de processamento, porém sem ganhos significativos de qualidade. Também se percebe pelos ensaios que o refinamento no nível de quarto de pixel traz pouco ganho de qualidade. O mesmo não procede para o refinamento de meio-pixel, que produz realmente um ganho significativo de qualidade quando habilitado (cerca de 0,9dB de melhoria da qualidade). Com base nesses dados, a topologia que desabilita-se o refinamento de quarto de pixel bem como o particionamento de vetores de movimento em blocos de 16x8 e 8x8 pixels mostrou-se a mais adequada para uma implementação rápida sem grandes perdas.

4.2.1.5 Experimento sobre Entropia

O padrão SVC permite a utilização do módulo de entropia CABAC, porém continua suportando o algoritmo CAVLC. Considerando essa questão, este novo ensaio buscou avaliar na prática qual o módulo de entropia mais adequado para uma implementação de baixa capacidade de processamento (OSORIO; BRUGUERA, 2004).

Foram realizados dois ensaios práticos: um utilizando o algoritmo CAVLC e outro o CABAC. Os resultados obtidos são apresentados na Tabela 34 (APÊNDICE A).

Os resultados de qualidade PSNR dos vídeos reconstruídos apresentam pequenas variações nas comparações. De fato, as variações registradas na tabela indicam uma relação indireta em relação ao efeito de cada algoritmo. Ou seja, o módulo de entropia, por operar apenas sobre o fluxo de saída, buscando uma melhor forma de codificar os valores válidos, não pode afetar a qualidade do vídeo gerado. Seu efeito prático é alterar a compressão do fluxo de saída.

A explicação das variações registradas se deve basicamente ao efeito causado pelo módulo de controle de taxa que, estando habilitado no software JSVM, atua continuamente sobre o módulo de quantização a fim de gerar um vídeo com taxa de saída constante. Assim, quando o módulo de entropia comprimir mais os dados de saída o módulo de quantização pode ter sua intensidade reduzida, ainda mantendo a taxa de saída prevista. Já quando o módulo de entropia mantiver compressão reduzida, o módulo de quantização, para manter a taxa de bits desejada, terá sua intensidade aumentada. Esta alteração na intensidade do módulo de quantização afeta a qualidade final do vídeo recuperado.

Assim sendo, as informações de qualidade do fluxo de vídeo gerado neste ensaio não serão consideradas como critério decisivo de comparação. Já as medidas de desempenho, confrontando os dois algoritmos em termos de tempo de execução, serviram para comprovar que o algoritmo CAVLC é mais rápido que o CABAC, com variação registrada de até 15% em alguns casos. A ilustração dos resultados de entropia obtidos é apresentada na Figura 46.

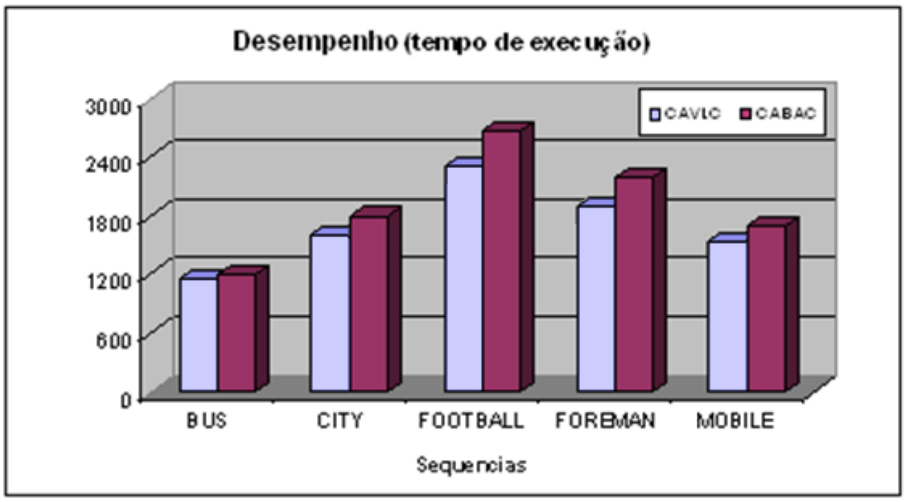


Figura 46: Comparação de desempenho referente módulos de entropia

Os resultados obtidos pela análise da complexidade computacional das duas metodologias também comprovaram que o algoritmo CAVLC apresenta uma complexidade menor que a necessária para a implementação do algoritmo CABAC.

Destaca-se na análise realizada o alto consumo de memória exigido pelo CABAC. A identificação visual desses resultados está apresentada na Figura 47.

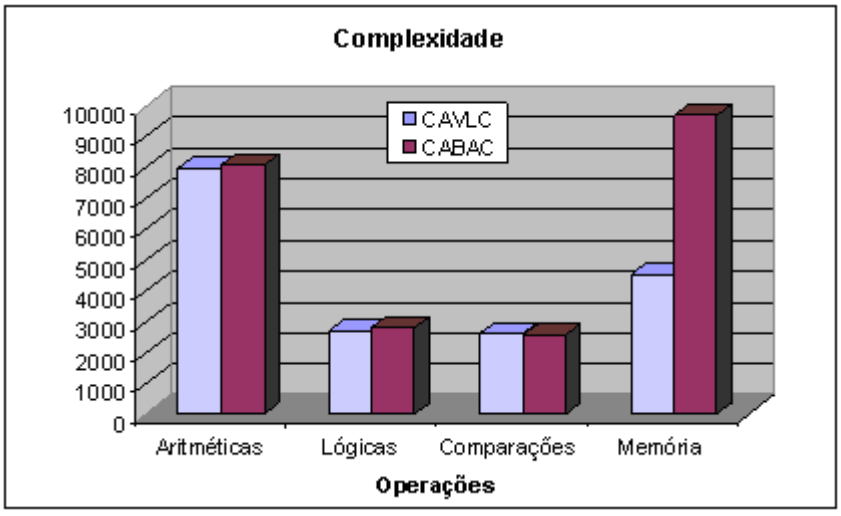


Figura 47: Avaliação da complexidade computacional dos módulos de entropia

4.2.1.6 Experimento sobre Filtragem Temporal

Outro recurso inovador da especificação SVC foi a inclusão de um filtro adicional de pré-processamento para aumentar a eficiência do algoritmo de escalabilidade temporal, o chamado filtro MCTF. A sua utilização, porém, não é obrigatória em uma codificação SVC. A título de verificação da influência desse filtro foram realizados alguns ensaios. Duas topologias foram ensaiadas: uma utilizando o filtro MCTF e outra não. Os resultados obtidos com estes ensaios são resumidos na Tabela 35 (APÊNDICE A).

A análise dos resultados aponta, em média, para uma melhoria da qualidade dos vídeos reconstruídos quando se utiliza o filtro MCTF, porém com resultados que são fortemente dependentes do tipo de movimentação presente no vídeo. De fato, com os resultados obtidos, fica difícil comprovar, de forma genérica, a real eficácia do filtro MCTF na melhoria da qualidade do vídeo gerado. Essa variação pode ser observada na Figura 48, referente ao elemento Y reconstruído pela camada 1.

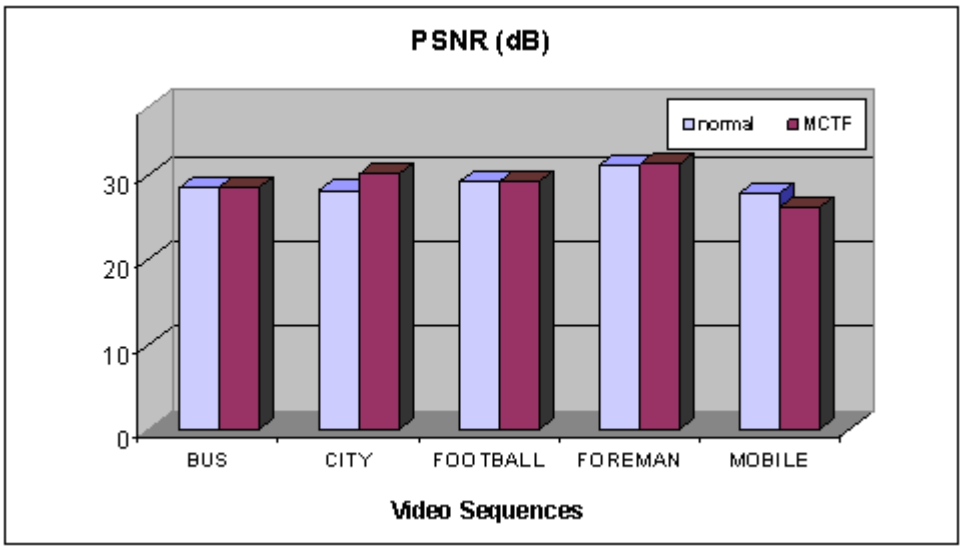


Figura 48: Comparação da qualidade da camada 1 referente ao módulo MCTF

A análise dos resultados de desempenho, entretanto, comprova a redução significativa de desempenho (aumento do tempo de execução) ao se utilizar o filtro MCTF, o que pode ser evidenciado na Figura 49.

Estes resultados obtidos vão ao encontro dos resultados de (SCHAFER et al., 2005), que desaconselha a utilização desse filtro para aplicações de baixa complexidade.

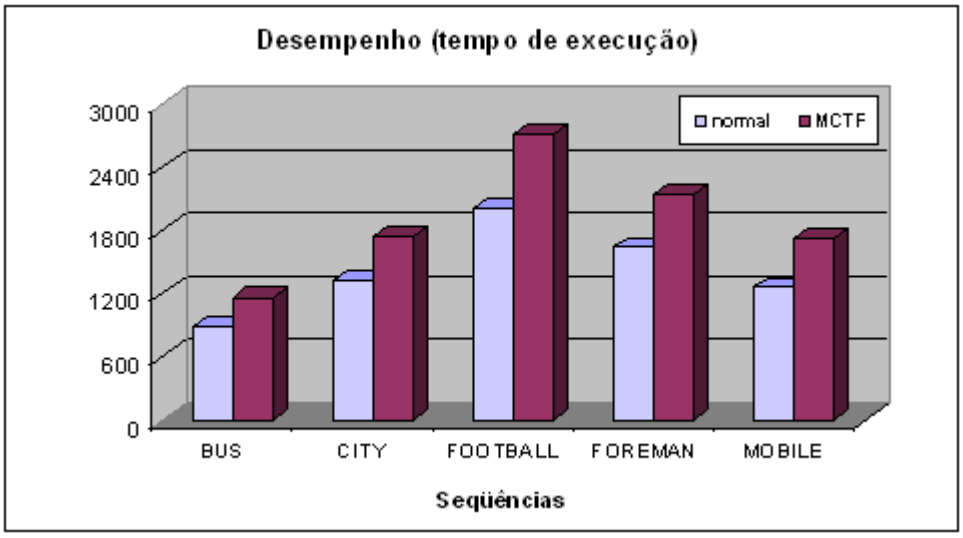


Figura 49: Comparação de desempenho referente ao uso do módulo MCTF

4.2.1.7 Refinamento do Modelo da Aplicação após Experimentos

Com base nestes ensaios pode-se avaliar o impacto prático causado por diferentes configurações de um codificador H.264/SVC. Estes resultados são apresentados de forma resumida na Tabela 2, onde se identifica para cada configuração os seus efeitos em termos de perda de qualidade, aumento de desempenho e redução do número de operadores.

Tabela 2 – Comparação de impacto para diferentes adaptações do H.264/SVC

Adaptação	Perda de qualidade	Aumento de desempenho	Redução no. operadores
Redução de GOP	0,2dB	10%	-
Remoção ESS (resize)	0,2dB	47%	17% (global)
CGS x MGS	1,2dB	5%	-
Predição: redução janela (24x24)	0,02dB	77%	-
Predição (sem 8x8 e 16x8)	0,05dB	65%	-
Predição (sem ¼ pixel)	0,1dB	45%	5% (global)
CAVLC x CABAC	0,8dB	20%	80% (memória)
Remoção MCTF	0,1dB	22%	15% (global)

Observando-se a tabela fica claro que a alteração de alguns algoritmos como o módulo de predição e filtro MCTF, podem ser modificados gerando ganhos significados de desempenhos com impactos muito baixos em termos de perda de qualidade final (menores que 0,1 dB). Outras configurações relacionadas diretamente os mecanismo de escalabilidade temporal (ajuste do GOP), espacial (ESS) e SNR (CGSxMGS), também afetam o desempenho e a demanda computacional (complexidade) dos codificadores, porém, para um adequado ajuste, devem levar em conta características desejadas da aplicação alvo.

Com base nestes resultados pode-se, de forma geral, definir uma solução refinada de codificador, que leva a aumentos significativos de desempenho, mantendo-se perdas de qualidade em níveis aceitáveis (menores que 0,5dB). Esta solução refinada foi assim considerada como aplicação alvo para o projeto de codificador escalável em questão.

Optou-se pela adoção de uma topologia de escalabilidade híbrida, que alia, em uma mesma solução, a baixa complexidade computacional de um codificador com escalabilidade temporal com a melhoria da qualidade final (vídeo reconstruído) obtida pela escalabilidade SNR. A escalabilidade espacial não foi implementada neste modelo visando a redução do tempo de execução, bem como a complexidade da solução. Cabe destacar que a escolha do tipo de escalabilidade a ser implementada pelo modelo levou também em consideração resultados de outras pesquisas acadêmicas. Em especial, pode-se destacar o trabalho de Daronco (2008), que realizou diversos ensaios práticos visando o levantamento da qualidade subjetiva de vídeos escaláveis. Os resultados obtidos apontam em uma melhor aceitação de vídeos que utilizam o conceito de escalabilidade SNR em detrimento dos que usam escalabilidade espacial. O modelo implementado, portanto, representa uma solução de escalabilidade híbrida que emprega as escalabilidades do tipo temporal e SNR.

Além da escolha do tipo de escalabilidade, outras simplificações foram implementadas no núcleo operacional do modelo refinado, a partir dos resultados práticos (Tabela 2).

O primeiro conjunto de simplificações definido diz respeito à redução de funcionalidades do módulo de predição, as quais, conforme medido, trazem pouco efeito na qualidade final, porém reduzem consideravelmente o tempo de execução. Mais particularmente, o módulo de predição refinado foi ajustado para trabalhar com refinamento apenas no nível de meio-pixel (não suportando assim a resolução de quarto de pixel). Além disso, restringe o recurso de particionamento de vetores de movimento (evitando-se assim a opção pelo uso de blocos como 8x8 e 16x8).

O módulo de MCTF, filtro de pré-processamento dos vídeos, que é sugerido pela norma SVC para aumentar a eficiência da codificação temporal hierárquica, não foi implementado no modelo refinado, por trazer poucos ganhos.

Por fim, optou-se por utilizar o mecanismo CAVLC como módulo de entropia do modelo, uma vez que resultados práticos comprovaram que o mecanismo CAVLC é mais simples e rápido que o CABAC.

Como resumo final destas definições, a Figura 50 traz uma representação simplificada deste modelo refinado de codificador escalável (aplicação alvo), apresentado na forma de diagrama de blocos. Pode-se observar como esta solução refinada segue a mesma estrutura apresentada na Figura 33 (codificador completo), porém de forma mais resumida.

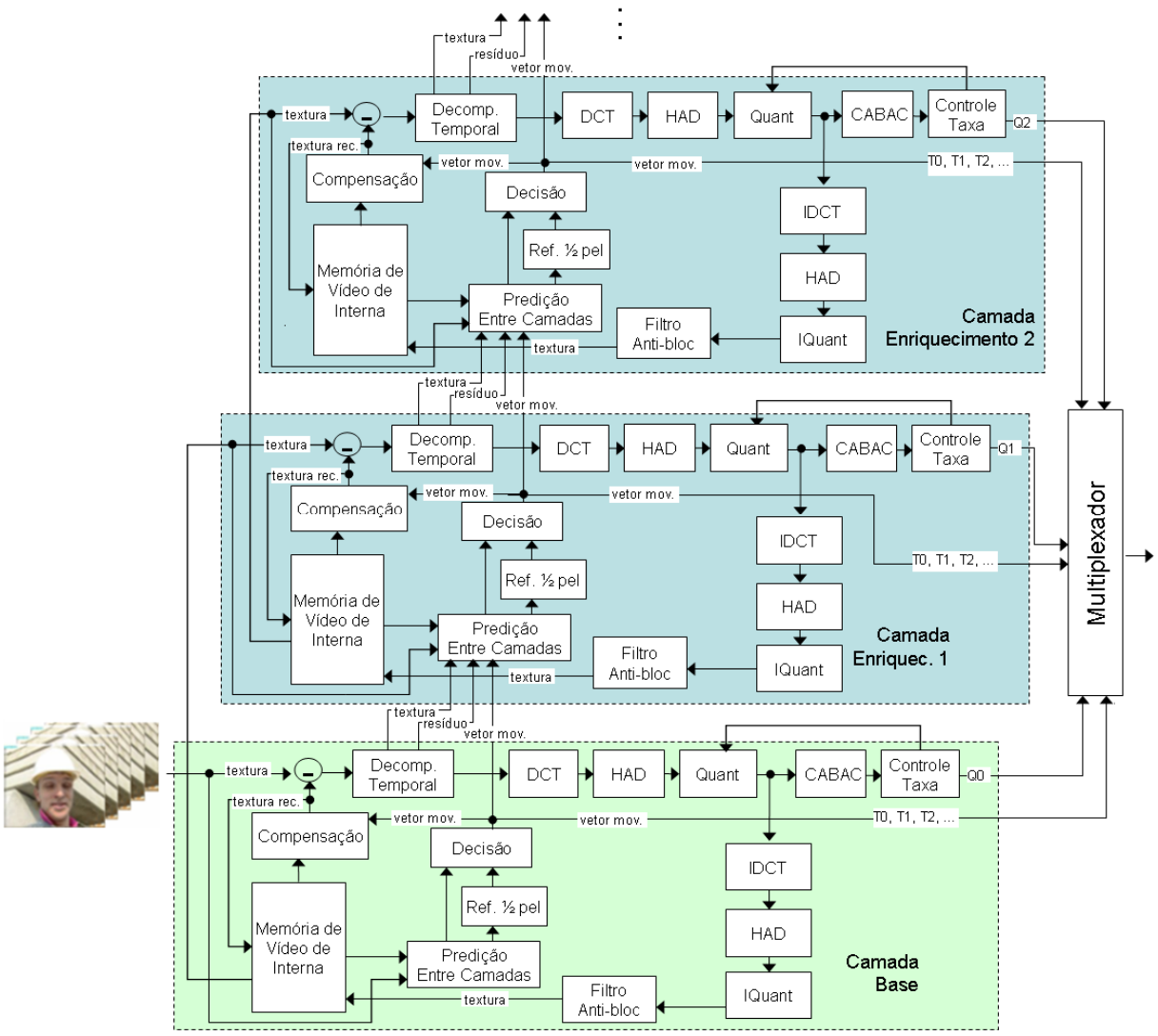


Figura 50: Diagrama de blocos do codificador escalável refinado

O exemplo ilustrado na Figura 50 representa uma aplicação escalável que implementa três camadas de qualidade (Q0, Q1 e Q2 respectivamente) e pelo menos três camadas temporais (T0, T1, T2 ...). Na prática o número de camadas de qualidade e temporais a serem adotadas deve ser flexível e configurável pelo usuário, devendo-se apenas respeitar a restrição do número máximo de 16 camadas de qualidade (versão MGS), segundo a especificação do codificador H.264/SVC.

4.2.2 Particionamento do Modelo Arquitetural

Projetistas de soluções tradicionais baseadas em ASIC têm uma visão centrada no hardware do sistema, ao mesmo tempo em que, analogamente, programadores têm uma visão centrada no software. A condição ideal para a etapa de particionamento é que os módulos sejam distribuídos, procurando colocar em hardware os algoritmos que apresentem maior potencial de paralelismo, de forma a produzir ganhos de desempenho (LAHTI et al., 2005).

Projetistas de ambos os sistemas devem analisar o desempenho em três dimensões: hardware, software e sistema. Na área do desenvolvimento de hardware, diversas características podem ser exploradas, tais como larguras dos operandos internos, organização de memórias, uso de pipelines, paralelismo de execução em diversos níveis, freqüências de operação, entre outras. No campo do software as explorações se processam considerando características como largura dos registradores internos, interfaces com memória, uso de operações de SIMD e técnicas de VLIW, distribuição de execuções em modo multi-tarefa, entre outras. Já no quesito sistema devem ser avaliadas questões práticas como modelo de integração entre componentes de hardware e software, sincronismo entre diferentes módulos, interfaces de comunicação, entre outras. As três dimensões (hardware, software e sistema) podem ser analisadas separadamente, porém na prática possuem forte inter-relação, sendo que uma decisão equivocada no desenvolvimento de uma delas pode afetar as demais. O ideal, neste caso, é que a definição do modelo arquitetural seja feita por um projetista que conheça bem todas as áreas envolvidas a fim de considerar as repercussões práticas das decisões tomadas (CHEN et al., 2009).

Durante o desenvolvimento do projeto em questão, procurou-se trabalhar com uma arquitetura modular, tanto para os componentes de software como para os de hardware, mantendo-se, assim, sempre que possível, as mesmas interfaces de entrada e saída.

A principal vantagem desta estratégia foi facilitar as etapas de depuração e integração dos diversos módulos desenvolvidos.

Conforme pode-se observar, analisando-se a Figura 50, a quantidade de módulos internos de um codificador escalável, mesmo depois de refinado é relativamente grande, tornando a tarefa de particionamento bastante trabalhosa. Entretanto pode-se observar que muitos dos módulos presentes entre camadas são similares, sendo apenas replicados entre as diferentes camadas, e por este motivo a análise pode ser feita apenas considerando-se os algoritmos de uma única camada.

Considerando-se esta estratégia, o número de módulos a serem analisados reduz consideravelmente.

Outra questão que pode ser observada é que muitos destes módulos internos não operam de forma direta com outros módulos vizinhos para em conjunto desempenhar uma determinada funcionalidade. Neste caso os módulos podem ser agrupados por funcionalidade (independentemente do mecanismo de funcionamento interno de cada um deles). Esta abordagem de agrupamento de módulos por funcionalidade também auxilia na alocação de tarefas entre hardware e software (CHEN et al. 2009) e por isso foi adotada no presente projeto.

De forma geral as seguintes funcionalidades foram previstas na forma de agrupamentos:

a) Processamento de dados de entrada: conjunto de módulos que realizam leitura e ajuste de formatos dos vídeos de entrada para se ajustar aos barramentos internos de operação;
b) Gerência do sistema: controle geral de funcionamento do codificador, administrando os fluxos de dados (textura, resíduo e vetor de movimento), dentro de cada camada ou mesmo entre camadas;
c) Módulo computacional intra: engloba principalmente os módulos de compressão direta por algoritmos de transformadas (DCT e HADAMARD) e quantização;
d) Módulo computacional inverso intra: Realiza a etapa de descompressão composta pelas transformadas inversas (IDCT e IHADAMARD) e quantização inversa;

e) Filtragem: módulo responsável pela atenuação de efeitos de blocagem nos vídeos reconstruídos;
f) Predição: engloba os módulos internos para compressão por correlação entre blocos (preditor entre camadas, refinamento de pixel e compensação);
g) Módulo de entropia: realiza a etapa de compressão por análise estatística dos dados que serão usados para compor o fluxo de saída;
h) Multiplexador: recebe os dados comprimidos das diferentes camadas a fim de gerar um fluxo escalável unificado;
i) Controle de taxa: módulo responsável por gerenciar o tamanho final do vídeo codificado, atuando sobre o módulo de quantização.

Com base nesta divisão de módulos pode-se considerar como modelo básico da arquitetura SVC a ser implementado, o diagrama de blocos apresentado na Figura 51, que representa, de forma simplificada, a estrutura adotada para codificação de uma única camada de acordo com a especificação H.264/SVC.

A Figura 51, além da apresentação dos principais módulos, já traz indicações de partes implementadas em software e hardware (partes em hardware apresentadas em caixas de cor sólida).

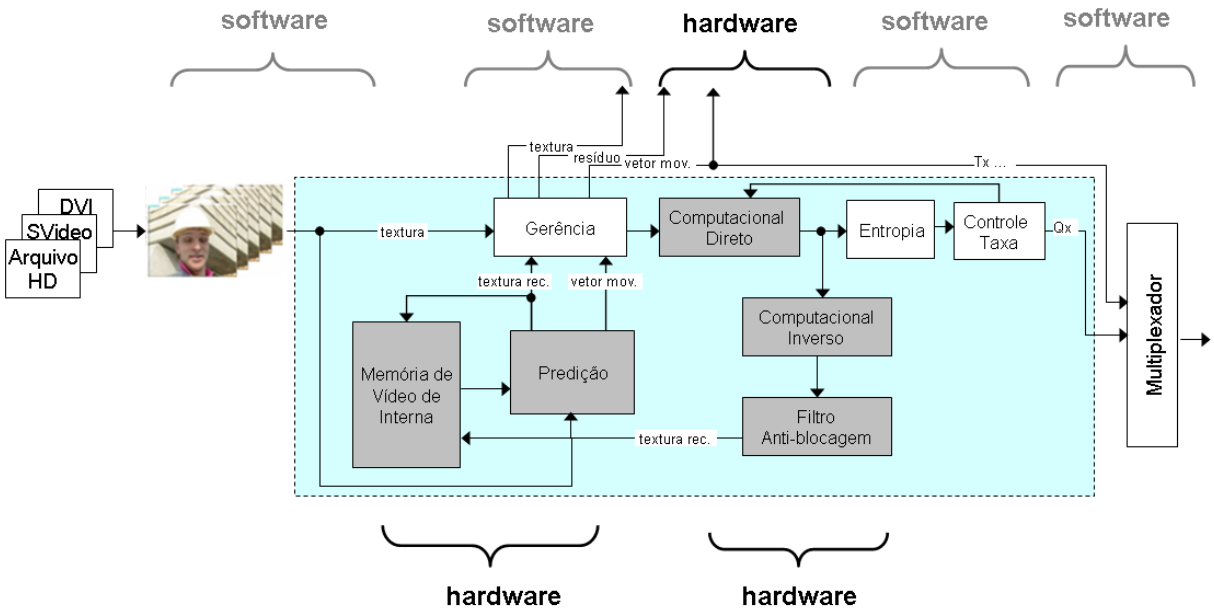


Figura 51: Diagrama de blocos de um codificador de vídeo H.264 de camada simples.

A descrição do funcionamento interno dos diversos módulos, bem como o detalhamento das decisões tomadas para particionamento deste, são apresentadas nas seções a seguir.

4.2.2.1 Módulo de Entrada do Sistema

O módulo de entrada de dados é responsável pela coleta de dados dos vídeos a serem codificados, que tanto podem vir de arquivo em disco rígido como de sinal de uma câmera. No caso de sinais de câmera devem ser observados os padrões de recepção (DVI, S-Video, HDMI entre outros) (KEITH, 2004). Sua funcionalidade depende de diversos fatores que devem ser definidos pelo usuário (tamanho do vídeo, formato de cores, número de quadros a serem processados, entre outros), normalmente a partir de um arquivo de configuração.

Independentemente de ser proveniente de arquivos em disco ou de uma câmera, sua leitura ocorre de forma seqüencial, com recebimento de um pixel por vez. Isso representa uma importante restrição do módulo. De fato, pode-se considerar que este módulo possui dependência de dados na ordem de unidade de pixel (ou seja, no melhor caso, um pixel seria processado por ciclo de execução).Além disso, por trazer a necessidade normalmente a necessidade de fazer interface com o usuário a partir de um computador, suas funcionalidades ficam bastante presas ao sistema operacional e por isso se torna uma tarefa pouco indicada para ser levada para uma plataforma de hardware . Sendo assim optou-se por manter este módulo como entidade de software no sistema desenvolvido.

4.2.2.2 Módulos Computacionais Direto e inverso

Os módulos computacionais, conforme identificado neste contexto, são responsável principalmente pelas operações matemáticas de compressão e descompressão de vídeo que englobam os procedimentos de transformadas e quantização. Estes módulos operam principalmente sobre os quadros do tipo intra, porém tem atuação também no processamento de resíduos proveniente de predição (intra ou inter quadro).

São dois tipos de transformadas propostas para um codificador H.264/SVC: transformada DCT e transformada Hadamard.

A implementação genérica da transformada DCT direta é dada pela multiplicação de cada amostra da matriz de entrada (luminância ou crominância) por uma matriz de coeficientes pela equação (RICHARDSON, 2003):

$$A_{ij} = A(i, j) = C_i \sqrt{\frac{2}{N}} \cos\left[\left(j + \frac{1}{2}\right)\frac{i\pi}{N}\right] \quad (1)$$

Onde: $A(i,j)$ representa cada valor que compõe a matriz de transformada;
i representa a posição do valor na direção horizontal;
j representa a posição do valor na direção vertical.

As constantes Ci são determinadas por:

$$\begin{cases} C_i = \sqrt{\frac{1}{N}} & (i = 0) \quad (2a) \\ C_i = \sqrt{\frac{2}{N}} & (i > 0) \quad (2b) \end{cases}$$

Onde: C_i representa o valor da constante na posição horizontal i;
N representa a dimensão da matriz de entrada.

Como se pode perceber, esta equação, por envolver operações de cosseno e raiz quadrada leva idealmente à necessidade de operações com lógica de ponto flutuante, o que aumenta a complexidade do algoritmo (TASDIZEN; HAMZAOGLU, 2005).

O codificador H.264, entretanto, utiliza uma expressão mais simplificada para implementar sua transformada DCT, que foi determinada alterando-se os coeficientes de transformadas a serem multiplicados por cada amostra de vídeo (componente de luminância ou crominância), para que sua implementação ocorra apenas utilizando-se operações de ponto fixo triviais (somas, subtrações e deslocamento de bits).

Com esta estratégia, se reduz a complexidade global do algoritmo (KORAH et al., 2008).

A operação completa de uma transformada DCT deve acontecer em duas dimensões, horizontal e vertical, o que é necessário por se tratar de imagens.

Assim sendo esta é normalmente implementada por duas operações de multiplicações de matrizes em seqüência: transformação direta e transformação transposta, como identificado na expressão a seguir:

$$Y = A.X.A^T \tag{3}$$

Onde: X representa o bloco de entrada de 4x4 pixels;
A representa a matriz de transformada direta;
A^T representa a matriz de transformada transposta.

A mesma abordagem é adotada durante a implementação da transformada Hadamard, que atua sobre as matrizes de saída da transformada DCT. Deve-se, entretanto destacar, que a matriz de entrada da Hadamard foi prevista para atuar, durante a codificação intra, apenas sobre os valores de primeiro coeficiente (componente DC) de cada um dos blocos de 4x4 amostras de um macrobloco (AMER; BADAWI; JULLIEN, 2005).

Já o algoritmo de quantização direta do codificador H.264/SVC é responsável por fazer uma redução controlada do número de bits de cada amostra, sendo definido pela equação a seguir:

$$Z_{ij} = round\left(\frac{|Y_{ij}|}{Q_{step}}\right) \tag{4a}$$

$$\text{Sign}(Z_{ij}) = \text{Sign}(Y_{ij}) \tag{4b}$$

Onde:
Z_{ij} é o elemento de saída quantizado;
Y_{ij} é o elemento de entrada (resultante das operações de transformada);
Q_{step} é o passo de quantização.

O valor de Q_{step}, nesse caso, é obtido a partir de tabelas que levam em consideração o parâmetro QP (*Quantization Parameter*), definido pelo usuário, e a posição de cada amostra dentro de cada matriz de 4x4 elementos.

A análise teórica destes módulos indica que os módulos de transformada operam a cada vez sobre blocos de 4x4 amostras. As operações exigidas, multiplicações de matrizes, podem ser trabalhadas de forma vetorial em linha ou coluna em momentos distintos (decorrente da necessidade de operações bidirecionais). Esse alinhamento dos algoritmos para operar com linhas e colunas torna possível o processamento paralelo de diversas amostras simultaneamente.

Já o módulo de quantização não apresenta dependência de dados específica, podendo operar em diferentes níveis de paralelismo (elementos a serem processados são totalmente independentes entre si).

Como, na estrutura interna de um codificador, o processo de quantização opera em série com os de transformada, o nível de paralelismo das transformadas pode ser estendido para o módulo de quantização de forma a garantir que o mesmo fluxo de dados da saída das transformadas também ocorra na saída da quantização.

Esta estratégia é especialmente importante para módulos ligados em série, garantindo uniformidade no alinhamento dos vetores transferidos de uma entidade para outra e buscando manter uma mesma vazão de dados entre todos os módulos internos.

Nesse caso, para o módulo computacional avaliado, pode-se considerar que a dependência mínima de dados para formar uma linha ou coluna de entrada é de quatro amostras. Esse nível de paralelismo exige barramentos de dados de 32 bits (quatro amostras de 8 bits), o que não chega a ser uma limitação significativa, por se tratar de um barramento comumente encontrado em plataformas de trabalho comerciais.

Vetores de dados maiores (por exemplo 64 bits) podem ser explorados, sempre, porém, observando-se as larguras de barramento disponíveis por memórias e também pelas interfaces de comunicação, a fim de garantir o fluxo efetivo de dados nas entradas e saídas de cada módulo (ZHAO; LIANG, 2006).

Com base nessas avaliações, o módulo computacional se mostra indicado para ser portado para hardware, considerando um nível de paralelismo mínimo, conforme avaliado, de quatro amostras.

4.2.2.3 Módulo de Filtragem

As especificações dos codificadores de vídeo H.264 (AVC e SVC) incluem um filtro adicional responsável por eliminar o efeito de blocagem normalmente presente na exibição de vídeos digitalizados (RICHARDSON, 2003).

A fim de implementar esta filtragem, devem ser avaliados a cada vez dois blocos de pixels vizinhos, inicialmente na direção horizontal e posteriormente na vertical. Estes blocos são divididos em sequências lineares de pixels, que são processadas individualmente. No caso

de filtragem horizontal essa seqüência de pixels é chamada de LOP (*Line of Pixels*) e no caso de filtragem vertical esta é chamada de COP (*Column of Pixels*). Uma ilustração dessa divisão é apresentada na Figura 52, onde ser observam dois blocos vizinhos (Bloco atual Q e Bloco prévio P) na direção horizontal:

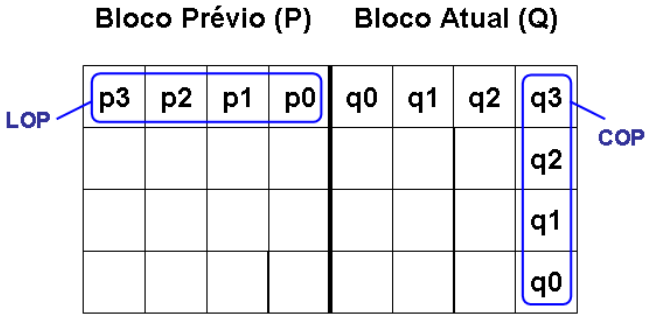


Figura 52: Divisão do bloco em linhas ou colunas de pixels para filtragem

Como pode-se observar na figura, cada LOP (ou COP) é composto por quatro amostras. No caso de uma etapa qualquer de filtragem, devem ser fornecidos ao filtro dois LOPs (ou COPs), respectivamente com a seqüência de pixels p3, p2, p1 e p0 do bloco anterior (bloco P) e pixels q0, q1, q2 e q3 do bloco atual (bloco Q).

Considerando essa distribuição o filtro a ser adotado, no pior caso, precisa apresentar uma dependência de dados de quatro amostras para formar uma linha ou coluna de um bloco de 4x4 pixels de um bloco passado, mais quatro amostras do quadro atual, gerando assim uma dependência de oito pixels para iniciar o procedimento de filtragem. Essa grande dependência de dados exige barramentos de 64 bits caso sejam buscadas implementações paralelas (MIN; CHONG, 2007). .

A dependência, entretanto, pode ser minimizada se forem utilizadas memórias internas para armazenar temporariamente informações de linhas ou colunas passadas, as quais serão posteriormente necessárias para a montagem do bloco prévio (bloco P). Assim apenas dados atualizados precisam ser fornecidos nas entradas do módulo. Esta estratégia aumenta a complexidade interna do módulo, mas, em compensação, evita a necessidade de novos acessos à memória global. Nesse caso a dependência de dados fica limitada a quatro amostras por etapa de filtragem, o que permite a sua utilização também em plataformas que adotem barramentos de 32 bits (LAI; CHEN; CHIOU, 2010).

Baseado nessa análise, que prevê um potencial de paralelismo compatível com o módulo computacional anteriormente analisado, o bloco de filtragem foi previsto para ser implementado também como um módulo de hardware, dentro deste projeto. Esta escolha traz

a vantagem adicional de permitir que toda a manipulação da memória de vídeo recuperado ocorra dentro da plataforma de hardware (etapa onde o processo de filtragem é, de fato, empregado), evitando assim trocas de dados adicionais entre módulos de software e hardware e, com isso, produzindo um efeito bastante significativo no aumento de desempenho do projeto colaborativo.

4.2.2.4 Módulo de Predição

Dentre todos os módulos de um codificador de vídeo H.264/SVC, o módulo de predição entre camadas é o mais complexo, devido a sua demanda de operar sobre grandes volumes de dados a cada laço de iteração para cálculo de correlação (GAO; DUANMU; ZOU, 2000).

Basicamente sua função é a de identificar a redundância temporal (predição inter) ou espacial (predição intra) entre quadros de vídeo, gerando informações resumidas de vetores de movimento (distância física entre dois blocos similares: atual e referência) e informações de resíduo (bloco resultante da diferença simples entre blocos similares). Posteriormente os blocos de resíduo são comprimidos pelo módulo computacional (NUNO; DIAS; SOUZA, 2005).

A implementação da predição entre camadas exige um algoritmo rápido de busca e cálculo de correlação entre blocos, seja para o caso de quadros de mesma camada ou mesmo entre quadros de diferentes camadas.

O algoritmo de busca deve percorrer várias posições relativas dentro de uma memória de referência a fim de localizar a posição do bloco mais similar dentro todos os pesquisados (BAE; THANG; RO, 2007).

Um diagrama de blocos genérico que ilustra este mecanismo está apresentado na Figura 53, onde se pode identificar a memória que armazena a área de busca (centro da figura), o módulo de cálculo de diferenças, o bloco de comparação (direita da figura) e o módulo de controle do mecanismo de estimativa de movimento.

A memória de referência (esquerda da figura) armazena os quadros de referência onde são pesquisados os blocos mais similares. No caso do codificador H.264/SVC estes quadros podem ser de quadros passados, futuros ou outra camada.

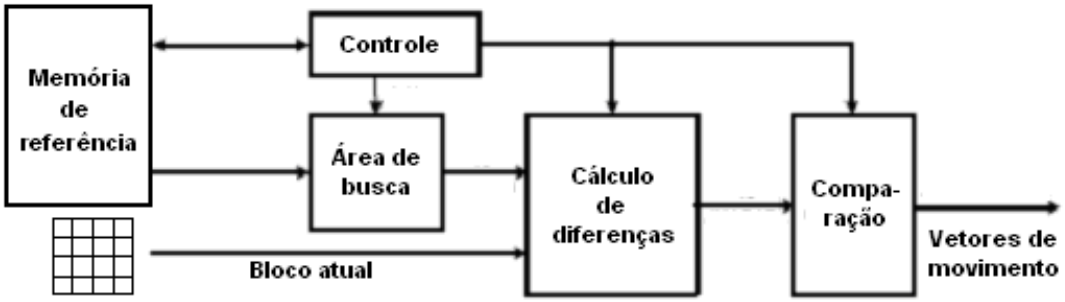


Figura 53: Diagrama de blocos genérico do mecanismo de estimativa de movimento.

O bloco de cálculo de diferenças é computacionalmente oneroso, pois para cada teste de similaridade deve ser computado o total das relações de diferença de todas as amostras do bloco atual com todas as amostras do bloco de referência pesquisado, o que pode levar a centenas de operações matemáticas por vez. Este valor é armazenado para comparação com os cálculos de diferenças de um bloco ligeiramente deslocado na memória de busca.

A seguir o resultado é comparado com outro bloco e assim vai sucessivamente acontecendo até que se localize por fim o bloco mais similar de toda a área de busca pesquisada (DAWEI; HOU; CHUNHUI, 2009).

Apesar, entretanto, de ser um algoritmo extremamente oneroso em termos de complexidade computacional, devido ao grande número de dados calculados a cada etapa de comparação, esse algoritmo é potencialmente sujeito à paralelização de operações.

Assim sendo, considerando a significativa importância deste módulo para o codificador, principalmente devido ao grande volume de dados envolvidos e ao elevado potencial de paralelismo desse algoritmo, sua implementação em hardware mostra-se plenamente justificável.

4.2.2.5 Módulos de Entropia e Controle de Fluxo

O bloco de entropia é responsável pelo tratamento final dos dados já codificados pelos blocos anteriores, a fim de gerar um fluxo de bits ainda mais reduzido.

A especificação H.264 inclui o algoritmo de entropia CAVLC, que se baseia no contexto atual do sistema, ou mais especificamente nos valores dos últimos elementos codificados (número de elementos diferentes de zero, elementos com valores 1 e -1, número

de zeros, entre outros), a fim de se gerar uma nova palavra de código mais otimizada. Este algoritmo consegue produzir um fluxo de bits mais comprimido que os obtidos por algoritmos de entropia tradicionais como o VLC (*Variable Length Coding*) empregados pelos codificadores MPEG-2 e MPEG-3, porém aumenta sensivelmente a dependência de dados (GHANDI; GHANBARI, 2003). Esta forte dependência de dados se deve principalmente a sua característica de fazer ajuste adaptativo de seus valores sintáticos (algoritmo orientado a contexto). Isso quer dizer que a próxima palavra de código somente pode ser calculada após a conclusão do código atual.

Além do algoritmo CAVLC, existe a possibilidade de se empregar alternativamente o algoritmo de entropia CABAC, que busca avaliar o fluxo de bits de saída como uma sequência binária relacionada, e com isso obter uma melhor forma de comprimir estes dados. A fim de se localizar a relação binária presente no fluxo de dados, estes devem ser interpretados sequencialmente bit a bit de forma adaptativa, causando uma dependência de dados em nível de bit. Este tipo de dependência torna o algoritmo bastante seqüencial e dificulta ainda mais a sua utilização em arquiteturas paralelas (LEE et al, 2006).

Já o módulo de controle de taxa deve ser responsável pelo ajuste dinâmico do tamanho ocupado pelo fluxo de bits de saída de cada camada, o que, na prática, é obtido pelo uso de um parâmetro de QP ajustado para cada camada.

Seu princípio de funcionamento se baseia, portanto, no ajuste controlado do parâmetro de quantização (elemento que atua sobre a distorção do sinal de saída) a fim de garantir uma taxa de dados efetiva na saída do codificador que atenda ao valor definido pelo usuário. Por este motivo, este módulo também é chamado de bloco de ajuste da taxa-distorção (*rate-distortion*).

Ambos os módulos operam serialmente com as amostras anteriormente codificadas pelo sistema, apresentando uma elevada dependência de dados inerentes aos seus procedimentos internos (análise baseada em contexto), e por isso apresentam um baixo potencial de paralelismo. Por esse motivo optou-se por manter esses módulos em software.

4.2.2.6 Módulo de Multiplexação

O módulo de multiplexação é responsável pelo processamento e montagem final do fluxo de saída. De forma geral, este módulo deve reconhecer e encapsular as diferentes camadas geradas a fim de produzir um fluxo de dados padronizado com o formato SVC, ou

seja, o enquadramento do fluxo de saída , segundo a definição de pacotes NAL (Network Abstration Layer) do padrão SVC (RICHARDSON, 2003).

Pode-se alternativamente configurar o codificador H.264/SVC para trabalhar com o recurso de compatibilidade AVC para a camada base, a fim de permitir sua exibição mesmo em um decodificador H.264 convencional (não escalável).

Além disso, por se tratar exclusivamente de um algoritmo que tem forte relação com o sistema operacional do computador, decidiu-se por manter a implementação deste módulo em software apenas.

Cada ainda salientar que justifica-se esta decisão pelo fato do algoritmo de multiplexação envolver o empacotamento de dados em um complexo formato de mensagens seriais, com baixa complexidade matemática e pouco susceptível à paralelização.

4.2.3 Integração dos Módulos

As primeiras iniciativas de co-projetos de hardware-software surgiram na década de 80 como soluções alternativas que integravam, em uma mesma placa, microprocessadores de propósito geral (parte de software) e componentes de propósito específico (parte de hardware) do tipo ASIC. A comunicação entre as diferentes entidades era feita por memórias compartilhadas ou registradores sendo estes os primeiros elementos a serem verificados para validar a proposta (DE MICHELI; EMST; WOLF, 2002).

A partir da década de 90, com a popularização da arquitetura aberta do computador pessoal da IBM (IBM-PC), começaram a se difundir soluções interoperáveis, ou seja, que poderiam ser interligadas em computadores de diferentes empresas. Isso, pois se passou a utilizar barramentos por slots como interfaces de comunicação (WOLF, 2003).

Baseado neste conceito, propõe-se uma solução onde a parte de hardware é implementada em uma placa com interfaces de barramento de alta velocidade.

A placa de hardware deve incluir componentes de tecnologia FPGA, onde devem ser desenvolvidos módulos próprios que rodam de forma colaborativa com uma solução de software.

A parte de software, por sua vez, deve operar em um computador pessoal que executa uma versão adaptada do código de referência H.264/SVC mantido pela entidade JVT para plataforma de PC (JOINT SCALABLE TEAM MODEL, 2010).

Na solução desenvolvida, o software de referência foi adaptado, de forma a se comunicar em tempo de execução com a placa FPGA, onde foram instalados módulos de hardware otimizados, a fim de acelerar o desempenho global da codificação H.264/SVC (Figura 54).

A integração entre os componentes de software e hardware ocorre por mensagens transmitidas através da interface de comunicação.

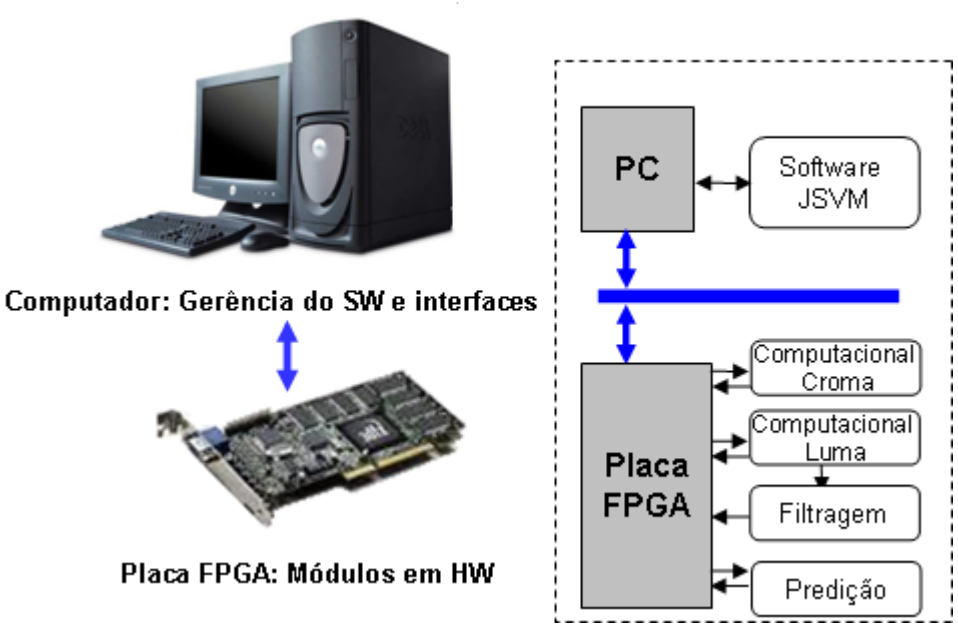


Figura 54: Representação dos módulos que compõe o projeto cooperativo proposto.

Baseando-se na versão refinada do código de referência H.264/SVC, foram criadas versões dos módulos em software, com as mesmas interfaces (programação modular), que, porém, ao invés de executar as operações diretamente no computador, montam pacotes de informações com os dados a serem codificados e enviam estes como mensagens para o hardware por chamadas ao *driver* de comunicação. Quando os dados tiverem sido processados pela placa serão então devolvidos e escritos nas respectivas estruturas de resposta.

Junto a cada mensagem enviada, são anexadas informações que servem para orientar o hardware nas suas tarefas de codificação, tais como tipo de dados de origem (luminância ou crominância), relação entre quadros (codificação intra ou inter), posição absoluta dos dados em relação ao quadro da imagem atual, número de camadas especificadas, qualidade desejada (QP) para cada camada, tipo de algoritmo requisitado, entre outros. Particularmente dentre essas informações o tipo de algoritmo requisitado é uma das informações mais importantes. É

essa informação, por exemplo, que define para qual módulo de hardware os dados devem ser enviados e conseqüentemente onde os dados de saída devem ser escritos.

A implementação feita em hardware da placa deve possuir uma máquina de estados principal, chamada de gerenciador de interface, que é responsável por processar todos os dados (mensagens) que vem do computador, identificando a funcionalidade requisitada. Na sequência deste módulo, são disponibilizadas diferentes memórias de entrada do tipo FIFO (*First Input First Output*). Cada uma destas FIFOs de entrada está relacionada com um agrupamento distinto de algoritmos implementados em hardware (computacional luma, computacional croma, predição entre camadas, entre outros). Da mesma forma, os dados de resposta destes módulos são colocados em FIFOs de saída dedicadas (Figura 55).

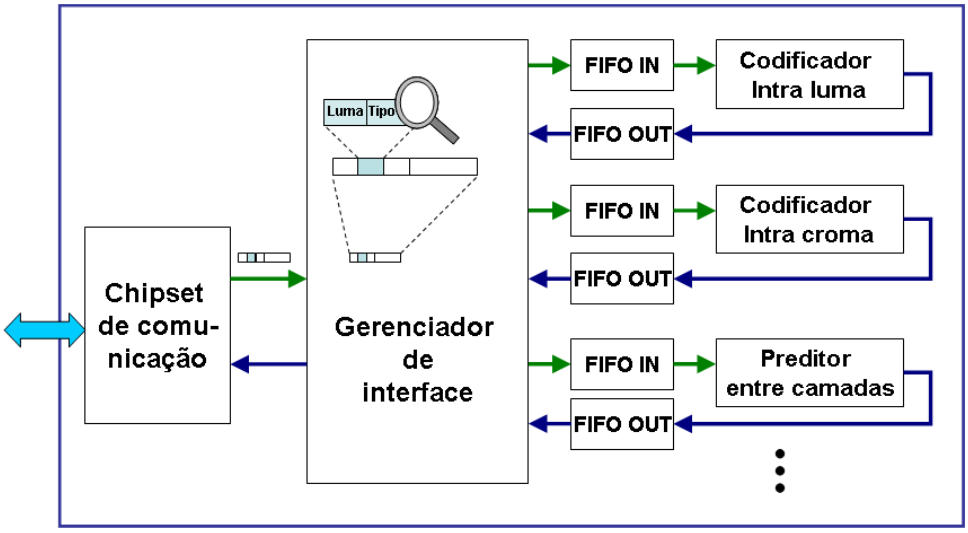


Figura 55: Representação da estrutura interna da placa na integração SW-HW.

A idéia desta abordagem é permitir o funcionamento independente e simultâneo dos distintos módulos em hardware de forma independente da interface de comunicação.

Uma questão relevante, entretanto, a se observar para esse tipo de projeto, é que a grande divergência entre características temporais de operação das diversas entidades envolvidas (freqüência da CPU, tempos da interface de comunicação, latência do sistema operacional e relógio da FPGA), dificulta a tarefa de sincronização entre os módulos.

Para estas situações, técnicas tradicionais tais como uso de memórias compartilhadas, envio de mensagens de sincronismo e utilização de semáforos não se mostram eficientes, principalmente, devido ao fato de que os módulos de software e hardware estão em plataformas de trabalho distintas com ligação dada por uma interface de comunicação assíncrona, que incorpora protocolos relativamente complexos (PCI ou PCI Express).

Alia-se a isso o problema do sistema operacional inserir atrasos adicionais cada vez que um acesso à interface de comunicação for requisitado (CHEN et al, 2009). Chamadas a essas interfaces, assim como demais aplicações que rodam em paralelo no computador, entram na fila do escalonador de tarefas do sistema operacional, que posterga seu atendimento por tempos variáveis, dependendo das demais tarefas na fila, o que, no final, acaba inserindo uma latência bastante significativa a cada mensagem (dados experimentais são apresentados na Seção 7).

Uma estratégia possível para essa situação é implementar um mecanismo de troca de dados do tipo bloqueante. Com essa estratégia, a mensagem de leitura pode ser requisitada logo após a de escrita. Quando o pedido de leitura chegar na placa de hardware e os dados não estiverem prontos, o hardware bloqueia a resposta até que os dados estejam de fato processados e atualizados em sua FIFO de saída. A desvantagem dessa técnica é o bloqueio da aplicação no computador (parte de software) cada vez que um acesso ao barramento externo é necessário, o que acaba restringindo o desempenho global do sistema.

Outra técnica é o uso do procedimento de varredura (*polling*) do estado da placa. Nesta técnica, o aplicativo, na entidade de software, requisita um serviço ao hardware através do envio de um pacote de dados. A seguir, o software faz chamadas ao hardware de informações do seu estado de operação. Quando o hardware terminar seu processamento, ele passa a sinalizar que concluiu a operação. Quando a entidade software reconhecer isto pelo processo de varredura, este poderá então coletar os dados já processados. Assim, o software não fica necessariamente bloqueado esperando o retorno do hardware, podendo desempenhar outras tarefas próprias entre uma varredura e outra. Apesar de aparentemente simples e funcional, na prática, esta técnica é diretamente influenciada pela latência do sistema operacional em atender aos pedidos de acesso ao hardware, gerando tempos significativos de espera que reduzem o desempenho do sistema (CHEN et al., 2009).

Visando-se aprimorar essa questão, pode-se adotar um mecanismo de agendamento de tarefas de leitura de mensagens de resposta. A idéia é que cada conjunto de dados enviado para a placa terá associado a si um tempo mínimo de espera (*deadline*). Assim, sempre que uma determinada funcionalidade for requisitada para a placa em hardware, logo após o envio dos dados, o tempo atual no sistema é registrado. Somando a esse valor o tempo total de execução do algoritmo na placa, tem-se assim o tempo mínimo a se esperar (*deadline*) para uma nova leitura desses dados já processados na placa (ATITALLAH et al., 2011).

A estratégia de agendamento de tarefas de leitura traz a maior flexibilidade para o projeto colaborativo, pois elimina a dependência entre as duas entidades (hardware e software), que, em alternativas anteriores, acabava limitando a possibilidade de execução simultânea das entidades. Entretanto, para que a técnica de agendamento de tarefas seja eficaz é necessário que se faça a determinação dos tempos corretos de resposta de cada serviço requisitado, os quais dependem de inúmeros fatores, tais como interface de comunicação utilizada, tamanho do pacote de dados processado, sistema operacional adotado, entre outros. Estes tempos mudam para cada ambiente de trabalho sendo necessário calibrá-los sempre que ocorrer uma troca de máquina ou configuração do ambiente.

Outra estratégia possível é o uso do conceito de interrupções. Esta técnica funciona de forma similar ao agendamento de tarefas, quando o aplicativo em software faz a requisição de serviços para a entidade de hardware e depois pode realizar outras tarefas enquanto este serviço estiver sendo processado. A diferença principal reside no fato de que o hardware pode avisar através de um evento de interrupção quando concluiu seu serviço. Uma rotina especial de tratamento de interrupção deve ser criada assim no aplicativo para atender ao pedido de interrupção feito pelo hardware e capturar da placa os dados processados. A rotina de tratamento de interrupção pode então atualizar diretamente as funções que necessitavam destes dados através de métodos de escrita direta nos buffers de trabalho ou apenas indicar através de variáveis de sinalização (*flags*), de que novos dados foram recebidos e estão disponíveis e assim as próprias tarefas podem buscar estes dados. Nesta estratégia as mensagens em hardware precisam indicar o tipo de serviço processados a partir de campo de dados específico ou a própria entidade em software descobre isso através serviço de leitura de estado da placa (CHEN et al., 2009).

4.2.4 Avaliação Teórica do Modelo

Dispositivos programáveis do tipo FPGA permitem grande flexibilidade para exploração de soluções paralelas, o que torna esta tecnologia ideal para implementação de algoritmos de alto desempenho. Entretanto devido à grande quantidade de pixels de dados, que devem ser processados em uma solução de codificação de vídeo, esta tecnologia requer o uso de memórias externas.

Na prática, alguns dos mais conhecidos gargalos de um sistema de codificação de vídeo embarcado estão relacionados à taxa de dados da memória e o mecanismo de comunicação entre a memória e a unidade computacional (YANG; WOLF; VIJAYKRISHNAN, 2005). Sendo a avaliação teórica e prática destas restrições parte fundamental do projeto, antes de qualquer implementação, dois requisitos específicos foram teoricamente avaliados nesta seção: (i) largura de memória da placa e (ii) taxa de bit da interface de comunicação (YE; MCGREGOR, 2008).

4.2.4.1 Avaliação das Características de Memória

Para a avaliação inicial da capacidade de memória, levou-se em consideração, como a situação de maior demanda de uso de memória, o caso de um codificador de vídeo SVC com algoritmo de qualidade MGS habilitado, o qual, segundo a especificação H.264/SVC, deve suportar um máximo de 16 camadas escaláveis. O maior consumo de memória previsto, para fins de verificação deste projeto, foi considerado quando todas as camadas ocupam a resolução máxima HD (1920x1080 pixels) (RIECKL, 2008).

Considerando que os dados de entrada adotem o formato 4:2:0, onde cada bloco de 4x4 amostras de componente de luminância corresponde a dois blocos de 2x2 amostras de componentes de crominância, se obtém um fator de multiplicação por 1,5. Assim, a quantidade de memória ocupada para esta aplicação escalável pode ser determinada como:

Camada HD : 1920x1080 = 2.073.600 pixels; (5a)

2073600*1,5 = 3.110.400 amostras por camada; (5b)

3110400*16= 49.766.400 amostras. (5c)

Ou seja, o total para todas as camadas é 49.766.400 amostras. Considerando uma resolução de 8 bits por amostra, chega-se à especificação de que a plataforma de hardware selecionada deve ter um requisito mínimo de memória de 50 Mbytes. Este é um requisito mínimo, pois só suportaria um quadro de referência. Para suportar um número maior de quadros de referência, a memória deveria aumentar proporcionalmente.

Além da capacidade da memória em si, é importante avaliar a taxa de comunicação com a memória a fim se determinar se esta tem capacidade compatível com a vazão máxima prevista.

A taxa máxima de acesso à memória tem a ver com a taxa de exibição escolhida para o vídeo. Assim sendo, considerando-se o caso anterior (16 camadas HD), para se operar com uma taxa de 30 qps (quadros por segundo), ter-se-ia a demanda máxima de 1,5Gbytes/s.

A freqüência de acesso da memória depende então do barramento de dados adotado, o que pode ser observado pela Tabela 3:

Tabela 3 – Demandas de comunicação com a memória externa

Demanda máxima	Barramento de dados	Frequencia (MHz)
1,5Gbytes/s (49.766.400 x 30)	32 bits	373
	64 bits	186
	128 bits	93

4.2.4.2 Avaliação de Desempenho das Interfaces de Comunicação

Além da avaliação da memória, deve-se também avaliar a interface de ligação entre hardware e software. Apesar de inicialmente a seleção da interface de comunicação parecer um detalhe menor, na prática, esta interface, quando mal escolhida, pode representar um importante gargalo para o sistema, ainda mais quando se pretende operar com vídeos de alta definição (CHEN et al., 2009).

Nas primeiras soluções de projetos colaborativos de hardware e software baseados em PC o principal barramento de comunicação era o ISA (*Industry Standard Architecture*), que permitia a comunicação paralela com a CPU principal com barramentos de 8 e 16 bits. Com a evolução crescente do mercado tecnológico este barramento acabou sendo condenado à obsolescência (YE; MCGREGOR, 2008).

Pode-se dizer que o sucessor do padrão ISA foi o barramento PCI (*Peripheral Component Interconnect*), criado em 1992, que também conectava distintos componentes do computador ao processador principal por meio de uma interface paralela. Em sua primeira versão utilizava um barramento de 32 bits com freqüência de 33 MHz (132 Mbytes/s), enquanto que a segunda versão suportava um barramento de 64 bits com freqüência de 66 MHz, que leva a uma taxa máxima teórica de 528 Mbytes/s, suportando leituras e escritas realizadas em modo rajada (XILINX, 2005).

Um exemplo típico deste tipo de interface pode ser encontrado na Figura 56, onde se observa dois dispositivos PCI em um mesmo barramento, que se conecta ao barramento principal através de um *chipset* próprio.

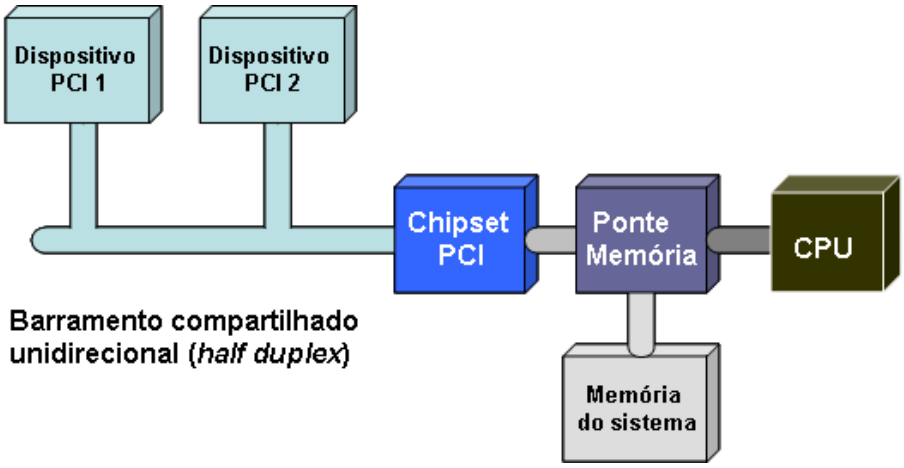


Figura 56: Estrutura interna do barramento PCI

Dois tipos de elementos podem figurar em um barramento PCI: inicializador e alvo. O inicializador é o dispositivo que inicia a transação, gerando sinais de endereço e controle. Mais de um dispositivo inicializador pode compartilhar o mesmo barramento, o que expande as flexibilidades de uso. Graças a todas estas características o barramento PCI passou a ser empregado por distintas soluções práticas de co-projetos HW-SW (MURACH et al., 2006).

Em 1996, a Intel lançou o barramento AGP (*Accelerated Graphics Port*), visando o mercado de placas de vídeo, que demandavam maiores taxa de dados. Trata-se de um barramento dedicado ligado diretamente ao *chipset* da placa-mãe, evitando assim o compartilhamento deste com outras placas do sistema e com isso aproveitando de forma mais efetiva o barramento. O barramento AGP permite a operação em modo rajada (assim como o barramento PCI), porém suportando a operação em paralelo de distintos canais para difusão de informação (também chamados de *lanes*). A versão mais simples AGPx1 suporta uma taxa teórica máxima de 266Mbytes/s, enquanto que a versão de 8 canais, AGPx8, suportava 2112Mbytes/s. Apesar das altas taxas suportadas este barramento não foi muito explorado para o desenvolvimento de soluções genéricas de projeto HW-SW cooperativo, basicamente por ser este dedicado para o mercado de placas de vídeo.

Para suprir esta ausência, foi lançada em 2003, a interface PCI Express, comumente abreviada como PCIe. Embora esta interface seja usualmente chamada de barramento no sentido estrito da palavra PCI Express não é um barramento, já que cada dispositivo conectado possui uma interface de comunicação exclusiva com o *chipset* da placa-mãe

(Figura 57), ao contrário do PCI, onde todos os dispositivos compartilham o canal ou barramento (AGILENT, 2006).

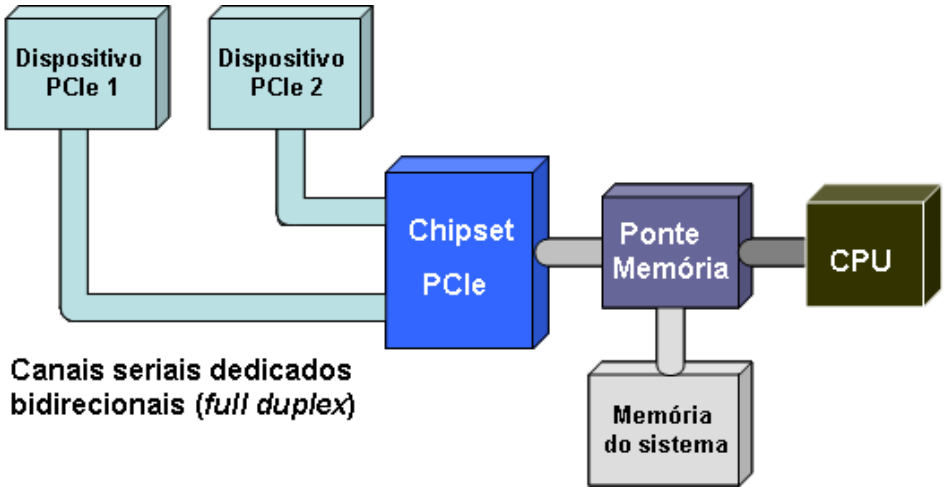


Figura 57: Estrutura interna da interface PCIe

Outra diferença essencial em relação aos seus antecessores, é que enquanto estes compartilham vias bidirecionais de 16, 32 ou 64 bits para estabelecer uma ligação paralela com o controlador de barramento. Já o *PCI Express* utiliza, em cada via, apenas dois bits unidirecionais, um em cada direção, perfazendo assim uma interface de comunicação serial.

No PCIe 1.1 cada via opera em uma freqüência de 2,5GHz. A cada byte serial transmitido são anexados dois bits de verificação e correção de erro. Desta forma cada via possui uma taxa teórica de dados de 250 Mbytes/s em cada direção. Um dispositivo PCIe pode possuir 1, 2, 4, 8, 16 ou 32 vias (PCIe x1, x2, x4, x8, x16 ou x32), permitindo-se que se possa alcançar taxas de dados de até 8GBytes/s em cada sentido (caso da PCIe x32).

Outras interfaces seriais, como USB e Firewire, foram criadas para interligação de dispositivos externos com a placa mãe, porém devido às limitadas taxas de dados disponibilizadas, estas não se mostram adequadas para aplicações que demandam volumes de informação muito elevados.

Outra alternativa é a interface Ethernet, que suporta taxas de até 10Gbits por segundo, porém traz como desvantagem o fato de ser um barramento eminentemente sem garantias temporais (não determinístico), o que dificulta o seu uso em aplicações de tempo real (CHEN et al., 2008).

Para a verificação do canal de comunicação considerou-se uma taxa de exibição de vídeo de 30 quadros por segundo e resolução de oito bits por amostra. Considerando-se

novamente o uso de vídeos com resolução HD, chega-se à quantidade máxima de dados transferidos para a placa a cada segundo como:

Camada HD : 1920x1080x1,5 = 3.110.400 amostras. (6a)

3.110.400*30 = 93.312.000 amostras/s (6b)

93.312.000*8 = 746.496.000 bit/s (6c)

A quantidade de dados de resposta da placa vai depender do algoritmo requisitado, pois alguns apresentam elevadas taxas de compressão (ex. predição de movimento) enquanto que outros apenas manipulam os dados sem alterar sua resolução em termos de bits (ex: filtro anti-blocagem). O pior caso seria para a situação em que a mesma quantidade de dados de entrada é retornada da placa. Considerando este caso a conclusão é de que o canal de comunicação precisa suportar um fluxo de pelo menos 1,5 Gbit/s (envio e resposta de dados).

A partir deste valor calculado como parâmetro de validação do projeto, a tabela a seguir resume as principais soluções técnicas disponíveis, indicando de antemão aquelas que podem ou não atendê-lo.

Tabela 4 – Análise dos principais canais de comunicação

Interfaces	Freqüência (MHz)	Largura (bits)	Taxa teórica máxima (Mbit/s)	Suporta aplicação alvo?
USB v2	480	1	480	Não
Firewire	400	1	400	Não
PCI v1	33	32	1066	Não
PCI v2	66	64	4266	**Sim**
PCI Express x1	2500	1	2000	**Sim**
PCI Express x8	2500	8	16000	**Sim**

É importante destacar que esta avaliação apresentada está analisando apenas o atendimento teórico das interfaces citadas. Na prática, atrasos decorrentes de compartilhamento de recursos, escalonamento de sistema operacional e sincronização entre hardware e software reduzem reduzir o fluxo efetivo de dados. Essas questões são avaliadas em maiores detalhes na Seção 6 (resultados experimentais).

5 MÓDULOS DE HARDWARE DESENVOLVIDOS

5.1 ARQUITETURA COMPUTACIONAL PARA CODIFICAÇÃO INTRA

A arquitetura para codificação intra representa uma solução que realiza a compressão do vídeo, considerando apenas informações internas de cada quadro. Nesse contexto, a arquitetura do codificador não inclui o módulo de predição entre quadros.

De forma geral este procedimento depende diretamente de operações de transformada, quantização e filtragem. Para facilitar sua identificação, neste trabalho, foi chamado de "módulo computacional" o conjunto de entidades responsáveis pelas operações de transformadas e quantização do codificador intra.

Por decisão de projeto, o módulo computacional da presente solução foi implementado em hardware, englobando quatro módulos principais, dois responsáveis pela compressão de dados (operações diretas) e outros dois módulos, responsáveis pela reconstrução do vídeo (operações inversas):

- Módulo de transformadas diretas;
- Módulo de quantização direta;
- Módulo de quantização inversa;
- Módulo de transformadas inversas.

Além do "módulo computacional", a especificação H.264 prevê ainda um mecanismo de predição intra como técnica adicional de compressão, identificando entre blocos já recebidos os mais similares, de forma a permitir o reaproveitamento dos mesmos. Uma apresentação detalhadas das diferentes entidades de hardware desenvolvidas é feita a seguir.

5.1.1 Módulo Computacional Direto

Em um processo de codificação de vídeo, os algoritmos de transformadas e quantização são utilizados como parte essencial do mecanismo de identificação de redundância espacial. Na prática, o princípio de funcionamento dos algoritmos de transformadas é o de mudar a representação dos dados originalmente do domínio espacial

para outro (domínio da transformada), o que normalmente, permite que estes possam ser representados com uma quantidade menor de valores válidos.

No algoritmo H.264/SVC são utilizados três mecanismos distintos de transformada, conforme a seguir listados:

- DCT 4X4;
- Hadamard 4x4;
- Hadamard 2x2.

5.1.1.1 Transformada Direta DCT

Quando habilitada a escalabilidade de qualidade, tipo de escalabilidade explorada no projeto em questão, a transformada DCT, do codificador H.264/SVC, deve ser aplicada sobre blocos de 4x4 pixels oriundos da entrada de vídeo (imagem capturada) quando se tratar da camada base, assim como blocos de 4x4 pixels resultantes do cálculo de resíduos, que são calculados e repassados para as camadas de enriquecimento.

Na prática, a transformada do codificador H.264 representa uma aproximação ortogonal da DCT original, de forma a utilizar apenas aritmética inteira (o algoritmo original exige notação em ponto flutuante).

Conforme a especificação H.264, o algoritmo DCT pode então ser implementado utilizando apenas operações de somas e deslocamentos de bits, eliminando a necessidade de multiplicações, um recurso que aumentava consideravelmente a demanda computacional dos módulos de transformadas.

Conforme já definido, a implementação de uma transformada DCT é dada pela expressão genérica de transformadas apresentada na equação (3) (Seção 4.2.2.2). Em uma transformada DCT tradicional as matrizes de transformada A e A^T são definidas respectivamente como:

$$\begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.271 & 0.271 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.271 & -0.653 & -0.653 & 0.271 \end{bmatrix} \quad \begin{bmatrix} 0.5 & 0.653 & 0.5 & 0.271 \\ 0.5 & 0.271 & -0.5 & -0.653 \\ 0.5 & 0.271 & -0.5 & -0.653 \\ 0.5 & -0.653 & 0.5 & 0.271 \end{bmatrix}$$

(a) (b)

Figura 58: Matrizes de uma transformada DCT 4x4 original (a) direta e (b) transposta.

Já para o codificador H.264, onde as transformadas DCT 4x4 foram adaptadas para trabalhar apenas com aritmética de inteiros, as matrizes de transformada A e A^T passam a ser definidas como (RICHARDSON, 2003):

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

(a) (b)

Figura 59: Matrizes de inteiros da transformada DCT 4x4 (a) direta e (b) transposta.

Pode-se observar que os valores destas matrizes variam entre -2 e 2 em representação inteira. Isso foi possível no padrão H.264, pois os valores de escala decorrentes do cálculo da DCT original (equação (1) da Seção 4.2.2.2), passaram a ser incorporados no estágio de quantização do codificador, tornando assim as matrizes do estágio de transformada mais simples. Na prática, o estágio de quantização de um codificador já exige operações de multiplicação (e divisão) durante seu processo de reescalonamento de valores. Assim a especificação H.264 alterou a tabela de valores da quantização de forma que estas pudessem absorver as escalas da transformada. Essa estratégia permitiu a redução da carga computacional das transformadas mantendo a mesma carga computacional no estágio de quantização

Considerando a importância do módulo de transformada DCT dentro de um codificador, várias soluções em hardware já foram propostas. Prasoon e Rajan (2009), por exemplo, sugerem uma arquitetura simplificada para implementar um módulo de DCT-2D em hardware, adotando uma abordagem de processamento serial. A solução foi implementada usando uma estrutura em *pipeline* de cinco estágios, que calcula separadamente cada um dos 16 resultados de um bloco de 4x4. Trata-se de uma solução bastante compacta (ideal para dispositivos que requeiram menor consumo de potência), mas que limita drasticamente o desempenho global.

Outras soluções procuram manipular em parte ou no todo as operações internas da transformada com vistas a otimizar implementações em hardware em termos de área ou desempenho global.

Neste conceito, uma adaptação possível é modificar a equação (3) para operar em um mecanismo de dois estágios, conforme identificado a seguir:

Estágio 1: $Y_1 = A.X$ (7a)

Estágio 2: $Y = A.\ Y_1^T$ (7b)

Onde: X representa o bloco de entrada de 4x4 pixels;
A representa a matriz de transformada direta;
Y_1 representa o resultado do primeiro estágio de transformada;
Y_1^T representa a matriz Y_1 transposta.

Esta estratégia é basicamente sugerida a fim de possibilitar o reuso do módulo de transformada, uma vez que, nessa solução, ambos os estágios se limitam à multiplicação da matriz de transformada A por outra matriz (entrada direta ou transposta do primeiro estágio).

O algoritmo, nesse caso, pode ser resumido como um procedimento de multiplicação de matrizes, transposição dos resultados obtidos e repetição do estágio de multiplicação. Com isso é possível se implementar uma transformação DCT-2D (DCT bidirecional por se tratar de imagens), a partir de dois estágios de transformação DCT-1D (DCT unidimensional, um estágio para a direção horizontal e outro para a vertical). Uma representação em diagrama de blocos desta técnica é apresentada na Figura 60:

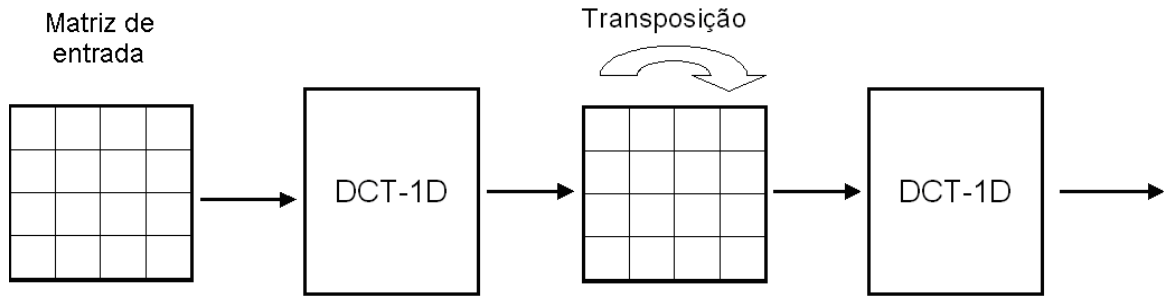


Figura 60: Procedimento de realização de uma DCT-2D a partir de estágios DCT-1D

Wang (2003) usa esta abordagem para implementar uma solução paralela baseada em dois módulos de transformadas unidimensionais interconectadas por um buffer de transposição. A solução proposta por Wang pode processar quatro amostras (amostras de luminância ou crominância) por ciclo de relógio (LI; HE; MEI, 2008).

Shirani (2005) adota uma abordagem inovadora, que explora propriedades de simetria apresentada na equação (1) a fim de produzir uma solução computacional sem uma etapa de transposição.

A abordagem proposta por Shirani pode processar até desesseis amostras por ciclo de relógio. Entretanto esta solução apresenta maiores atrasos no caminho crítico, devido ao fato de utilizar estruturas computacionais, sequencialmente mais complexas, o que também limita o desempenho geral.

Outra solução que explora paralelismo em hardware é apresentada por Agostini (2006), onde uma estrutura dedicada de somas e deslocamentos é proposta para realizar o processo de transformada DCT bidimensional de um bloco inteiro de 4x4 amostras seguindo uma estrutura em *pipeline*.

A alta eficiência computacional em termos do número de amostras por ciclo de relógio (uma matriz de 16 amostras é processada por vez), entretanto, impacta em restrições para atingir freqüências muito elevadas, uma vez que estruturas complexas normalmente impactam no aumento dos atrasos de operação (mínimo período da solução).

Além disso, muitas vezes limitações práticas das plataformas de trabalho podem afetar o desempenho da solução. Neste sentido, durante o desenvolvimento do projeto em questão, além da análise de arquiteturas de outros autores, foram observadas algumas questões práticas, tais como a largura do barramento de dados de memórias e interfaces de comunicação (32 ou 64 bits).

Assim, foi desenvolvida uma solução própria, que se utiliza de duas abordagens distintas: (i) arquitetura em pipeline para cada DCT-1D de 32 bits, a qual visa reduzir complexidade de cada operação e, por conseqüência, o período de trabalho, e (ii) estrutura modular duplicada, que permite a expansão direta da solução de 32 bits para suportar acessos a memórias de 64 bits sem mudanças na arquitetura do projeto.

Como pode-se observar pelas matrizes da DCT (Figura 60) a implementação em hardware de uma transformada DCT-1D é relativamente simples, uma vez que se limita a operações de soma e multiplicação por 2 (deslocamento de 1 bit para a esquerda).

Cada linha da matriz de entrada exige a realização de quatro expressões matemáticas (uma para cada elemento da linha). Entretanto por serem operações similares, pode-se aproveitar a simetria da matriz para se gerar uma solução de dois estágios, utilizando-se uma arquitetura de pipeline.

Para ilustrar isso, a Figura 61 apresenta a estrutura interna da solução desenvolvida para a transformada DCT-1D com um pipeline de dois estágios, utilizado para realizar o cálculo simultâneo de uma linha da matriz de entrada (4 amostras).

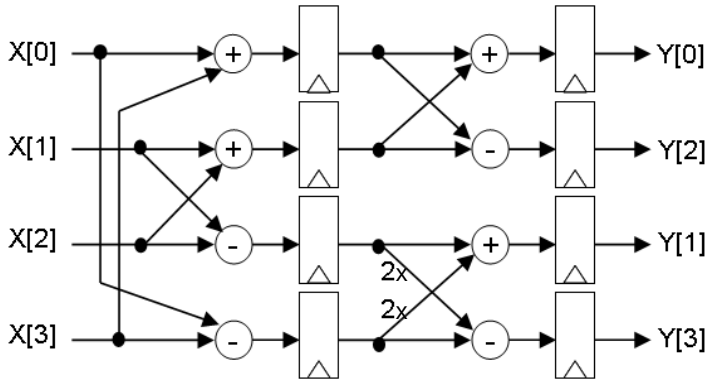


Figura 61: Implementação em hardware de cálculo de uma linha de uma DCT 4x4

Na prática, a operação completa de uma transformada DCT-1D demanda quatro etapas (uma para cada linha). Assim, ao se adotar uma solução em pipeline deve-se garantir um fluxo de quatro amostras por ciclo de relógio na entrada do módulo. Após a realização do primeiro procedimento de transformada (DCT-1D), a matriz de dados de saída deve ser transposta para então passar por outro procedimento de transformada DCT-1D.

Visando garantir alto desempenho para a solução, o procedimento de transposição é implementado por buffers de registradores, que possuem interconexões de entrada alinhadas em linha e saída em coluna. Quatro ciclos de relógio são necessários para preencher o buffer para, a partir daí, serem disponibilizados de forma transposta na saída.

Com base nessas considerações, foi desenvolvida uma solução de DCT-2D composta por dois módulos de hardware distintos:

- DCT-1D H, que realiza a transformada DCT na dimensão horizontal, juntamente com a transposição horizontal para vertical;
- DCT-1D V, que realiza a transformada DCT na dimensão vertical.

Uma visão geral desta arquitetura é apresentada na Figura 62.

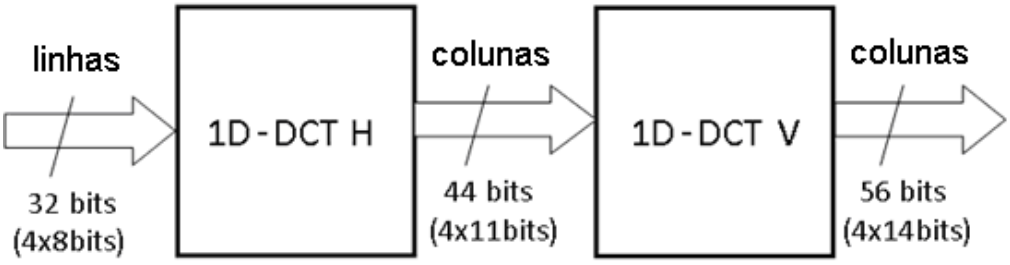


Figura 62: Solução para implementação de uma DCT-2D

É importante observar que as larguras de bit de entrada e saída são diferentes para cada módulo. Estas diferenças se explicam pelo fato das operações de somas e deslocamentos de bit aumentarem a resolução de bit das amostras. Em teoria pode-se acrescentar até três bits para cada operação de transformada, resultando em uma saída final de 14 bits por amostra. Na prática as operações computacionais internas de ambos os módulos DCT-1D H e DCT-1D V são muito similares, diferindo basicamente nas larguras de bit envolvidas.

Considerando estes estágios, a DCT-1D H tem um total de seis estágios (dois da transformada + quatro da transposição) de latência enquanto que a DCT-1D V ocupa apenas dois estágios, como pode ser observado na Figura 63.

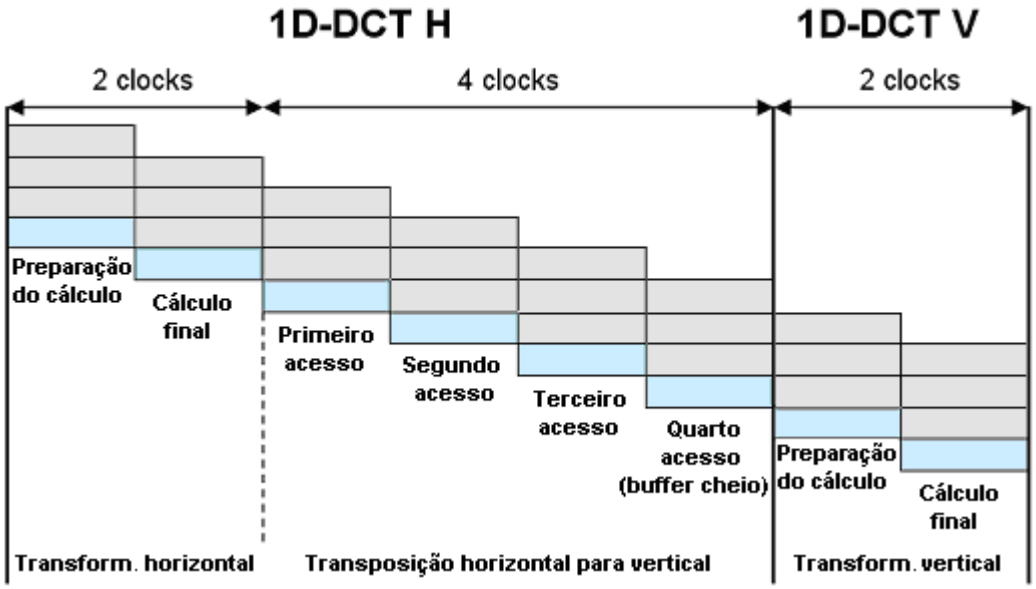


Figura 63: Descrição interna das operações para realização de uma DCT-2D

A solução desenvolvida consome e processa quatro amostras por ciclo de relógio, produzindo valores de saída na mesma taxa de entrada, após passar um período de latência total de oito ciclos de relógio (seis ciclos para DCT-1D H mais dois para DCT-1D V).

O processamento de quatro amostras de oito bits por ciclo de relógio é especialmente adequado para barramentos de 32 bits (4 x 8 bits), o que não chega a ser uma restrição crítica, pois pode ser encontrado em diferentes interfaces de comunicação e memórias comerciais. Entretanto, caso a plataforma de trabalho suporte, podem ser utilizados barramentos de 64 bits, que permite o acesso a oito amostras simultâneas. Para estes casos a arquitetura pode ser ajustada simplesmente a partir da duplicação de módulos internos.

A definição do ordenamento de dados nesses barramentos maiores levou em consideração que equipamentos de captura comerciais como câmeras geram suas informações de pixel orientadas sequencialmente em linha. A ordenação desses pixels começa do primeiro pixel (posição mais a esquerda da linha superior da imagem) até o último pixel desta linha (posição mais a direita). A seguir surge o primeiro pixel da segunda linha até o último pixel da mesma linha. Esse procedimento se repete linha a linha até a última linha da imagem.

Quando barramentos orientados à memória de 64 bits são usados, cada endereço pode armazenar um conjunto de oito amostras consecutivas, onde, deve-se adotar a mesma organização em linhas de pixel usada em dispositivos comerciais (Figura 64).

Nota-se que, nessa organização, cada endereço contém duas linhas consecutivas de blocos vizinhos de 4x4 pixels (caso de uma DCT-2D).

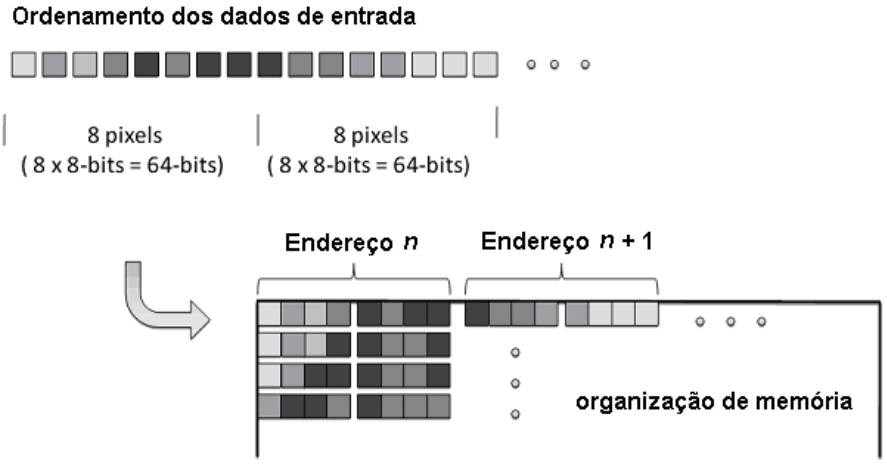


Figura 64: Organização convencional de pixels na memória.

Considerando esse fato, a arquitetura desenvolvida foi assim ajustada para receber e processar simultaneamente dois vetores de quatro pixels consecutivos de uma mesma linha (ou seja, relacionados a blocos 4x4 vizinhos). Nesse sentido o módulo DCT-2D foi internamente paralelizado, processando os 32 bits mais significativos (linha de quatro amostras do primeiro bloco) em paralelo com os 32 bits menos significativos (linha de quatro amostras do segundo bloco) cada vez que um novo vetor é recebido.

No estágio final (latência de oito ciclos de relógio), esta arquitetura duplicada produz simetricamente um vetor expandido de 112 bits, correspondendo a uma concatenação de valores internamente processados (8 x 14 bits por amostra processada).

Uma representação interna desta solução completa é apresentada na Figura 65.

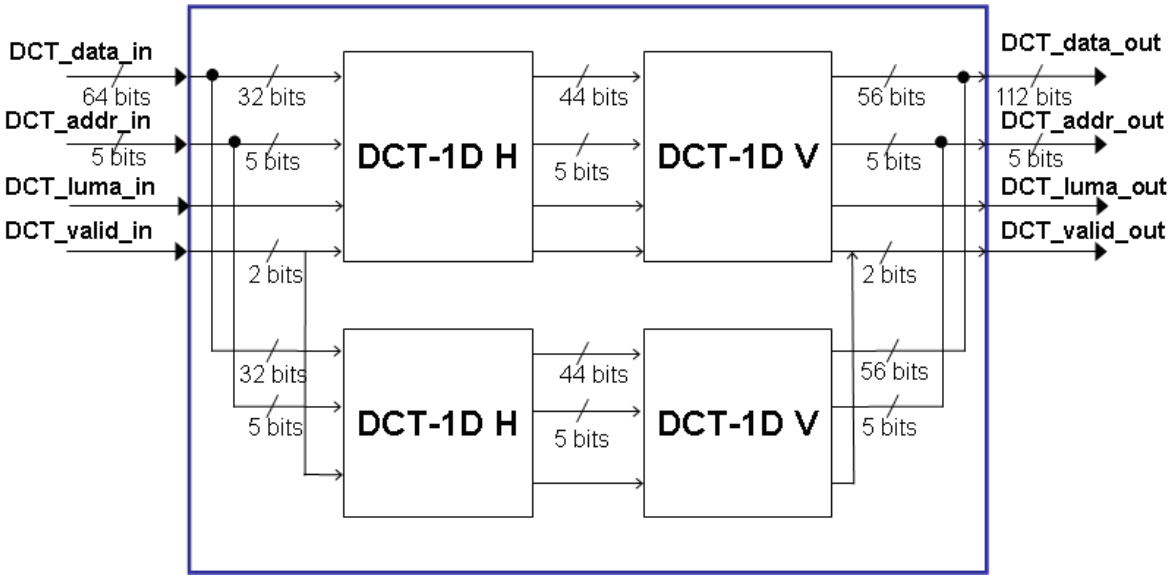


Figura 65: Estrutura interna da arquitetura DCT-2D duplicada para operar com 64 bits

Basicamente seus principais sinais implementados são:

- *DCT_data_in*: linha de valores de entrada, simultaneamente referenciando-se a dois blocos de dados. Cada amostra é representada por 8 bits;
- *DCT_addr_in*: identificador do número da linha que está sendo informado na porta anterior (*DCT_data_in*);
- *DCT_luma_in*: identificação de tipo de dado. Possui valor alto para componentes de luminância e valor baixo para componentes de crominância;
- *DCT_valid_in*: sinal que indica a presença de dados válidos na entrada do módulo;
- *DCT_data_out*: linha de valores gerados pelo processamento do módulo. Cada valor de saída é representado por 14 bits;
- *DCT_addr_out*: identificador de qual linha de saída está sendo gerada na saída *DCT_data_out*;
- *DCT_luma_out*: identificação de tipo de dado de saída. Possui valor alto para componentes de luminância e valor baixo para componentes de crominância;
- *DCT_valid_out*: sinal que indica a presença de dados válidos na saída.

Conforme pode-se observar, esta arquitetura foi desenvolvida para receber, associada aos dados de entrada, a posição particular de cada bloco 4x4 dentro da estrutura global de um macrobloco de 16x16 pixels (sinal *DCT_addr_in*). A utilização deste sinal de controle

simplifica a lógica interna da solução, pois elimina a necessidade de contadores internos para controle do estágio de transposição. A Figura 66 apresenta a ordenação adotada para cada amostra interna de bloco de 4x4 pixels. Cada um destes dezesseis blocos 4x4 (blocos 0 a 15) é composto por quatro linhas de amostras, que podem estar relacionadas com informações de luminância ou crominância. Na prática, a presença de luminância ou crominância não afeta a operação DCT-2D.

Assim as portas DCT_lin_num e DCT_col_num quando contém o valor “00000” indicam a primeira linha (ou coluna) dos blocos 0 e 1 enquanto que o valor “11111” representa a quarta linha (ou coluna) dos blocos 14 e 15.

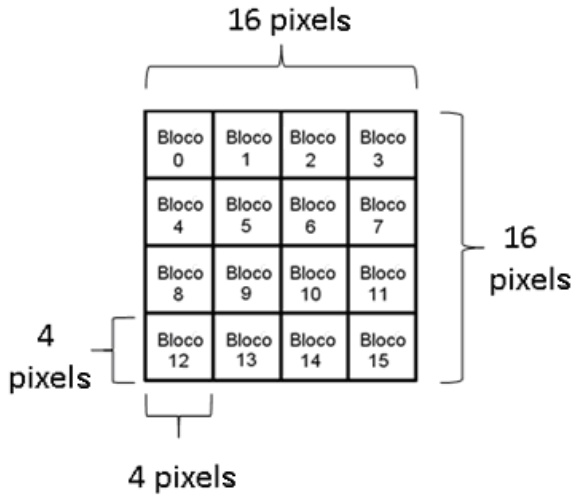


Figura 66: Ordenação de blocos dentro de um macrobloco

5.1.1.2 Transformada Direta Hadamard

A fim de reduzir o erro de reconstrução de imagens no processo de transformada DCT, o codificador H.264 especifica a utilização de um segundo tipo de transformada, a transformada Hadamard.

Esta transformada deve operar sobre os blocos de 4x4 elementos resultantes da transformadas DCT, porém não sobre todos os elementos, mas sim apenas sobre os coeficientes DC (primeiro componente resultante da transformada de cossenos).

A cada conjunto de 16 elementos (matriz de 4x4 pixels) tem-se um coeficiente DC. Assim, tomando-se a matriz resultante das transformadas DCT sobre um macrobloco inteiro de luminância (16x16 pixels) pode-se montar um bloco de 4x4 pixels de coeficientes DC.

Este bloco de componentes DC montado passa então por uma transformada Hadamard 4x4. Já se o macrobloco for de crominância (8x8 pixels), o bloco de componentes DC montado deverá passar por uma transformada Hadamard 2x2.

Uma representação desta montagem é apresentada na Figura 67 (formação dos blocos de coeficientes DC para cálculo da transformada Hadamard).

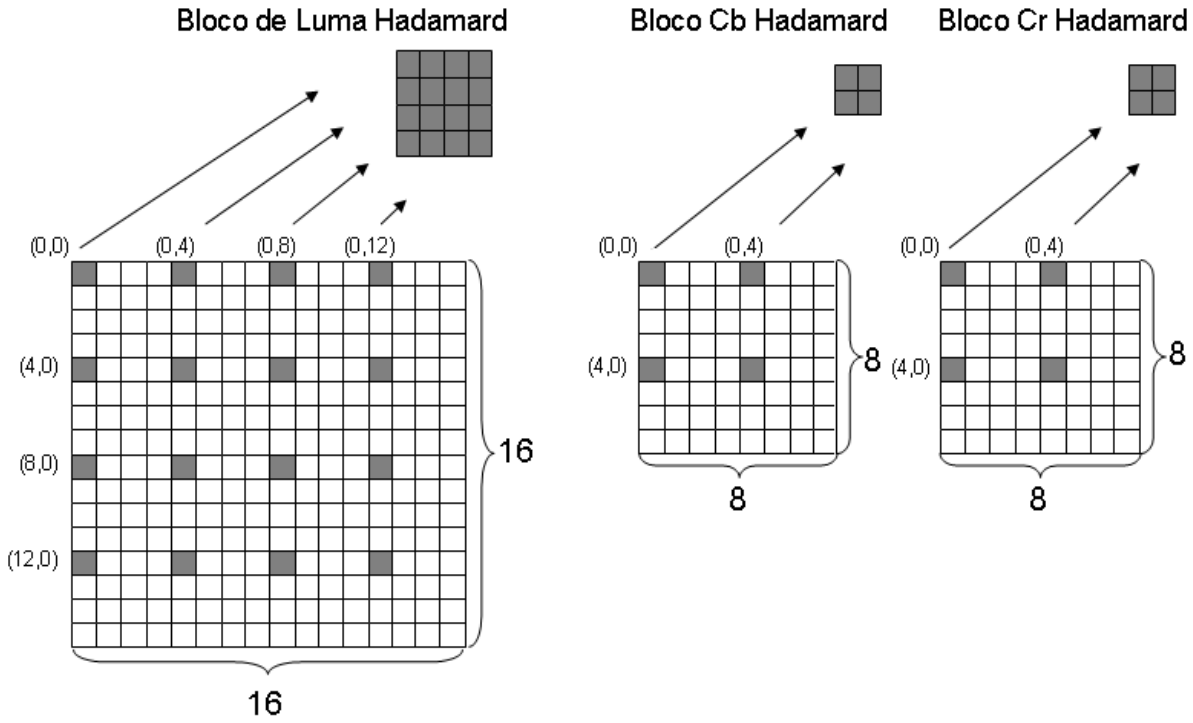


Figura 67: Montagem dos blocos de coeficiente DC para a transformada Hadamard

De forma resumida, a equação (5) ilustra a expressão genérica do cálculo da transformada de Hadamard. Como pode-se perceber o princípio de funcionamento da Hadamard é basicamente o mesmo das demais transformadas regidas pela equação (3), apresentada na seção 4.2.2, com a diferença apenas da necessidade de uma operação de divisão por dois, no estágio final do procedimento, a qual usada para ajustar os resultados finais do módulo:

$$Y = (H.X.H^T)/2 \tag{8}$$

Onde: X representa o bloco de entrada;
H representa a matriz de transformada Hadamard direta;
H^T representa a matriz de transformada Hadamard transposta.

5.1.1.2.1 Transformada Direta Hadamard para Componentes de Luminância

O tamanho das matrizes de coeficientes Hadamard está diretamente relacionado com as dimensões dos blocos de DC obtidos do módulo de DCT. Nesse caso, quando um macrobloco de luminância (16x16 elementos) for processado, as matrizes de coeficientes têm dimensões de 4x4 elementos, sendo compostas pelos valores apresentados na Figura 68:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

(a) (b)

Figura 68: Matrizes de inteiros da transformada Hadamard 4x4 (a) direta e (b) transposta.

Pode-se observar que, pela simetria da matriz de coeficientes *H*, as duas matrizes da transformada Hadamard *H* e H^T são iguais. Além disso, demandam apenas operações de soma e subtração, o que facilita ainda mais a implementação em hardware (LI, 2008).

A solução desenvolvida adota também uma estrutura de *pipeline* de dois estágios para a implementação da transformada Hadamard de 4x4 pixels, conforme pode ser observado na Figura 69:

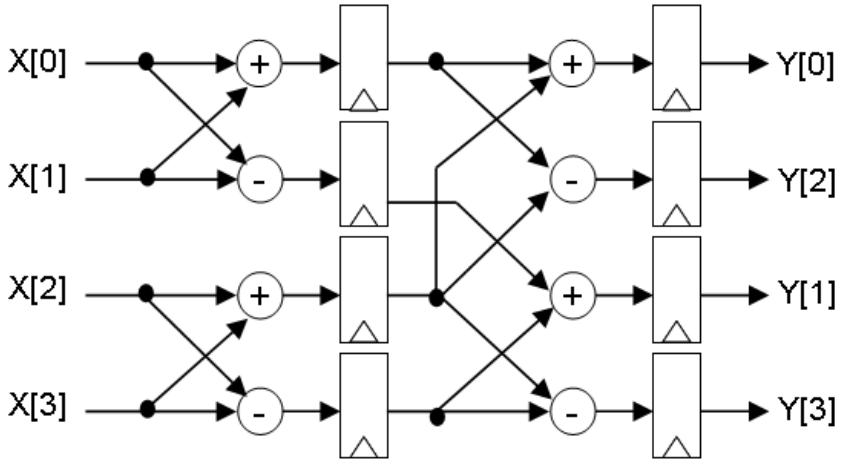


Figura 69: Implementação em hardware de cálculo de uma linha de uma Hadarmard 4x4

Assim como acontecia com a transformada DCT, para uma transformada Hadamard completa são necessárias quatro operações (uma operação para cada linha). Considerando-se

as barreiras temporais de cada operação a realização completa da transformada Hadamard 4x4 pode ser implementada também utilizando um *pipeline* de quatro estágios.

A operação de divisão por dois que deve ocorrer no estágio final de saída da transformada é implementada simplesmente desprezando o bit menos significativo dos resultados de saída, o que afeta apenas na lógica de interconexão do circuito, sem ocupar, portanto, células lógicas adicionais.

5.1.1.2.2 Transformada Direta Hadamard para Componentes de Crominância

Considerando-se o caso das informações de crominância, quando são montados blocos de DC de 2x2 pixels também, nesse caso, as matrizes de transformada H e H^T aplicadas sobre blocos 2x2 são iguais, sendo definidas como:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figura 70: Matriz de coeficientes da transformada Hadamard 2x2

Devido aos tamanhos envolvidos, uma implementação em hardware desta transformada é bem mais simples que as anteriores. Neste sentido, a solução de Hadamard 2x2 desenvolvida adota uma estrutura em pipeline de dois estágios, projetada para operar diretamente sobre toda a matriz de entrada (2x2 elementos = 4 amostras), conforme apresentado na Figura 71:

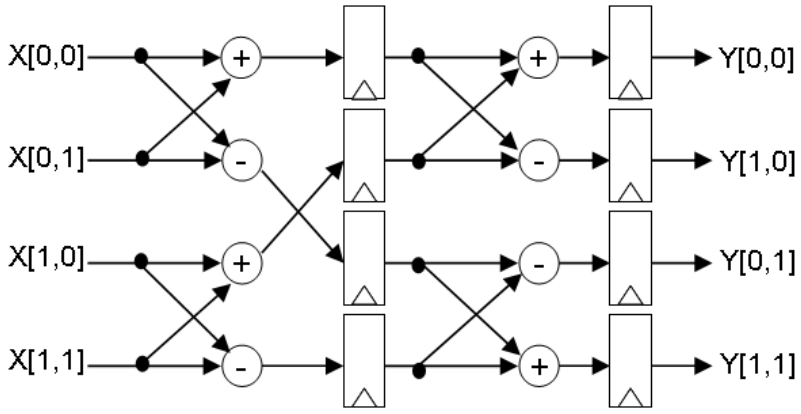


Figura 71: Implementação de uma transformada Hadamard 2x2 em hardware

5.1.1.2.3 Transformada Direta Hadamard 4x4 Completa

Na solução Hadamard completa proposta ambas as implementações para luminância (4x4)e crominância (2x2) foram incluídas nos mesmos módulos, visando-se reduzir a área total ocupada (a mesma lógica de controle e operadores internos puderam ser utilizados para ambos os casos). Nesta solução completa, o modulo de transformação Hadamard desenvolvido também adota a abordagem de duas etapas, interconectando dois módulos unidimensionais consecutivos (HAD-1D H e HAD-1D V respectivamente) para implementar uma transformada Hadamard bidimensional completa (HAD-2D).

Similarmente ao algoritmo de DCT as operações usadas são relativamente simples (somente somadores, subtratores e deslocadores de bit foram necessários). Entretanto estas operações incrementaram a resolução das amostras de 14 para 16 bits, gerando assim um barramento de saída de 128 bits (16 bits por amostra).

Dois buffers de memória internos são usados para permitir a mesma taxa de dados de entrada e saída (Figura 73). O buffer MB é responsável por armazenar os coeficientes originais não processados, enquanto os coeficientes DC são coletados e processados pelos módulos de cálculo, sendo, a seguir, e armazenados em um segundo buffer (buffer HAD) para posterior uso. O último módulo, chamado de HAD Comp, é responsável por acessar ambos os buffers, sincronizando os valores lidos para compor o barramento de saída Hadamard.

O buffer MB suporta 384 amostras (16 blocos 4x4 de luma e 8 blocos 4x4 de croma), enquanto o buffer HAD suporta 24 amostras (um bloco 4x4 de luma e dois blocos 2x2 de croma).

A Figura 72 apresenta o diagrama de blocos da solução Hadamard completa desenvolvida. De forma geral, os principais sinais implementados para essa solução são:

- *HAD_data_in*: linha de valores de entrada, simultaneamente referenciando-se a dois blocos de dados. Cada amostra de entrada é representada por 14 bits;
- *HAD_addr_in*: identificador do número da linha que está sendo informado na porta anterior (*HAD_data_in*);
- *HAD_luma_in*: identificação de tipo de dado. Possui valor alto para componentes de luminância e valor baixo para componentes de crominância;
- *HAD_valid_in*: sinal que indica validade de dados na entrada do módulo;
- *HAD_data_out*: linha de valores gerados pelo módulo (16 bits por amostra).

- *HAD_addr_out*: identificador de qual linha de saída está sendo gerada;
- *HAD_luma_out*: identificação de tipo de dado na saída. Possui valor alto para componentes de luminância e valor baixo para componentes de crominância;
- *HAD_valid_out*: sinal que indica validade de dados na saída do módulo.

.

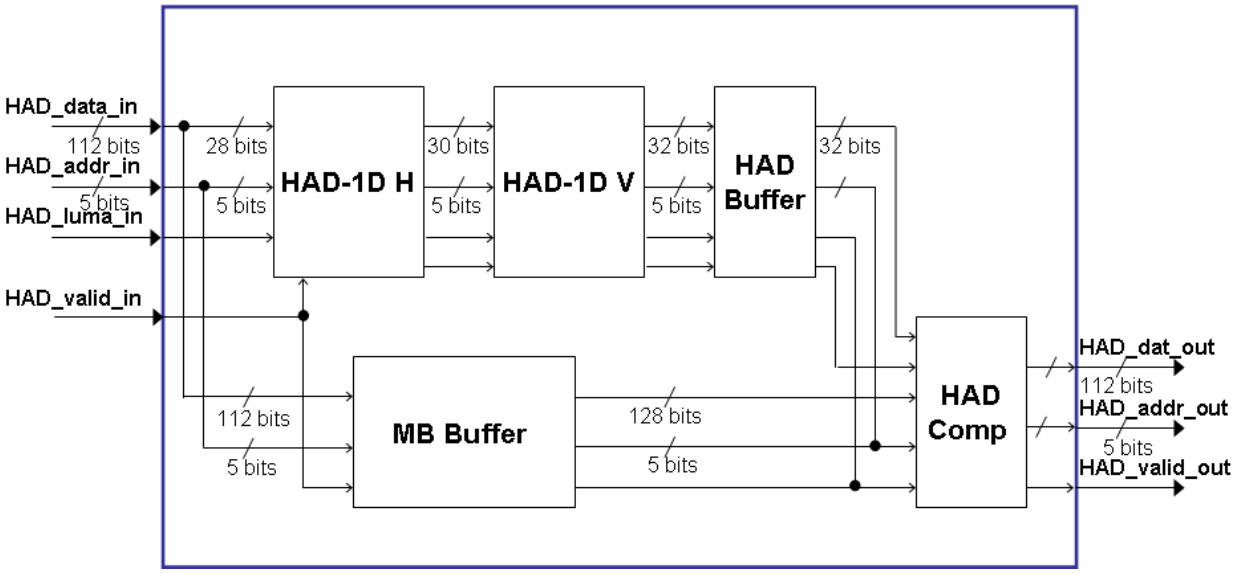


Figura 72: Estrutura interna do módulo HAD-2D

5.1.1.3 Módulo de Quantização Direta

O processo de quantização é responsável por reduzir a largura de bits dos dados codificados, como estratégia para comprimir o volume de dados gerado.

Nos quantizadores escalares, isso é implementado basicamente pela operação de divisão do valor original por uma variável de escala ou passo de quantização, que é obtido a partir de uma tabela de valores inteiros, conforme descrito pelas equações (4a) e (4b), apresentadas na Seção 4.2.2.2.

Na norma H.264 o passo de quantização (Q_{step}) deve variar de 0,625 to 224, dependendo do parâmetro de quantização chamado de *QP*.

A relação entre Q_{step} e *QP* é dada pela Tabela 5. Pode-se observar que os valores de QP variam de 0 a 51, propiciando uma faixa de 52 valores de Q_{step}, o que garante um controle bastante preciso da banda ocupada pela codificação de vídeo.

Tabela 5 – Determinação do valor de passo de quantização

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	...
QStep	0,625	0,6875	0,8125	0,875	1	1,125	1,25	1.375	1,625	1,75	2	2,25	2,5	...
QP	...	18	...	24	...	30	...	36	...	42	...	48	...	51
QStep		5		10		20		40		80		160		224

Fonte: Richardson, 2003

Deve-se lembrar que a tabela de quantização do codificador H.264 absorve os fatores de escala da transformada DCT. Esta estratégia foi implementada pela norma a fim de tornar o cálculo da transformada mais simples, reduzindo a complexidade global do codificador.

Assim sendo, o mecanismo de quantização do H.264 precisa levar em conta, além de QP (parâmetro que regula a intensidade da quantização) também a posição de cada elemento individualmente dentro da matriz de entrada (existe um elemento de escala adequado para cada coordenada *i* e *j* da matriz).

A fim de simplificar também a aritmética do módulo de quantização, a norma H.264 adaptou a equação genérica de um processo de quantização (equação 4a), de forma a utilizar para cada elemento apenas uma operação de multiplicação e um deslocamento de bit, eliminando assim qualquer operação de divisão. Assim sendo, a equação da quantização definida pela norma H.264 é identificada segundo a equação (KEITH, 2004):

$$Z_{ij} = round\ (\ (Y_{ij}\ .\ MF\ )\ /\ 2^{qbits}\ ) \qquad (9)$$

Onde:

Z_{ij} é o elemento de saída quantizado;
Y_{ij} é o elemento de entrada (resultante das operações de transformada);
MF é o fator de multiplicação que depende de *QP* e da posição do elemento de entrada na matriz;
qbits representa o número de bits a serem deslocados.

O valor de *qbits* é dado pela expressão abaixo:

$$qbits = 15 + floor\ (QP \text{ div } 6) \qquad (10)$$

O valor de *MF* foi ajustado na norma para que o valor da divisão seja sempre um número inteiro em potência de dois. Assim a operação tradicional de divisão pelo passo de quantização se limita a uma simples operação de deslocamento de bit, sendo, portanto mais adequado para implementações em hardware.

Na prática em hardware a implementação do processo de quantização pode ser representada por:

$$|Z_{ij}| = SHR\ (|Y_{ij}|.\ MF + f,\ qbits) \tag{11a}$$

$$Sign\ (Z_{ij}) = Sign\ (Y_{ij}) \tag{11b}$$

Onde:

SHR() é uma operação de deslocamento de *qbits*;
Z_{ij} é o elemento de saída quantizado;
Y_{ij} é o elemento de entrada (resultante das operações de transformada);
MF é o fator que depende de *QP* e da posição do elemento na matriz;
f é um valor definido como $\frac{2^{qbits}}{3}$ para blocos intra e $\frac{2^{qbits}}{6}$ para inter.
Sign() representa o sinal da amostra;

A determinação de *MF*, conforme definido anteriormente depende de *QP* e da posição do elemento a ser quantizado na matriz de entrada. Seu valor é dado pela tabela:

Tabela 6 – Determinação do valor numérico do fator de multiplicação do quantizador

QP	Posições (0,0) (2,0) (2,2) (0,2)	Posições (1,1) (1,3) (3,1) (3,3)	Outras Posições
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

Fonte: Richardson, 2003

Observa-se que, apesar do parâmetro QP variar de 0 a 51, na prática o algoritmo também trabalha com o fato de que o passo do quantizador duplica a cada seis incrementos no valor de QP. Assim, a complexidade da implementação reduz, considerando-se que a norma H.264 deve operar com módulo de 6 de QP. Ou seja:

$$MF_{QP>5} = MF_{QP = QP \bmod 6} \tag{12}$$

O processo de quantização para os coeficientes DC gerados após a transformada Hadamard difere um pouco da expressão de quantização anteriormente citada.

De forma resumida a expressão é definida como:

$$|Z_{ij}| = SHR\ (|Y_{ij}|.\ MF(0,0) + 2.f,\ qbits) \quad (13a)$$

$$Sign\ (Z_{ij}) = Sign\ (Y_{ij}) \quad (13b)$$

Onde:

SHR() é uma operação de deslocamento de *qbits*;
Z_{ij} é o elemento de saída quantizado;
Y_{ij} é o elemento de entrada (resultante das operações de transformada);
MF é o fator que depende de *QP* e da posição do elemento na matriz;
f é um valor definido como $\frac{2^{qbits}}{3}$ para blocos intra e $\frac{2^{qbits}}{6}$ para inter;
Sign() representa o sinal da amostra.

Os valores de MF, f e qbits são definidas de acordo com as informações de QP_mod (QP%6), QP_div (QP/6) e a posição (i, j) específica do elemento dentro da matriz de entrada. Na solução desenvolvida os cálculos de MF, f, QP_mod e QP_div foram implementados por tabelas em memória, reduzindo assim a complexidade computacional.

Barreiras temporais devem ser utilizadas para interligar os diferentes módulos a fim de se produzir uma arquitetura em pipeline. De fato, esta abordagem é vantajosa por aumentar o desempenho da implementação (KORAH et al., 2008). De forma geral, o processo de quantização desenvolvido implementa um módulo paralelo, que permite em sua versão de 32 bits o processamento de quatro amostras simultâneas (Figura 73).

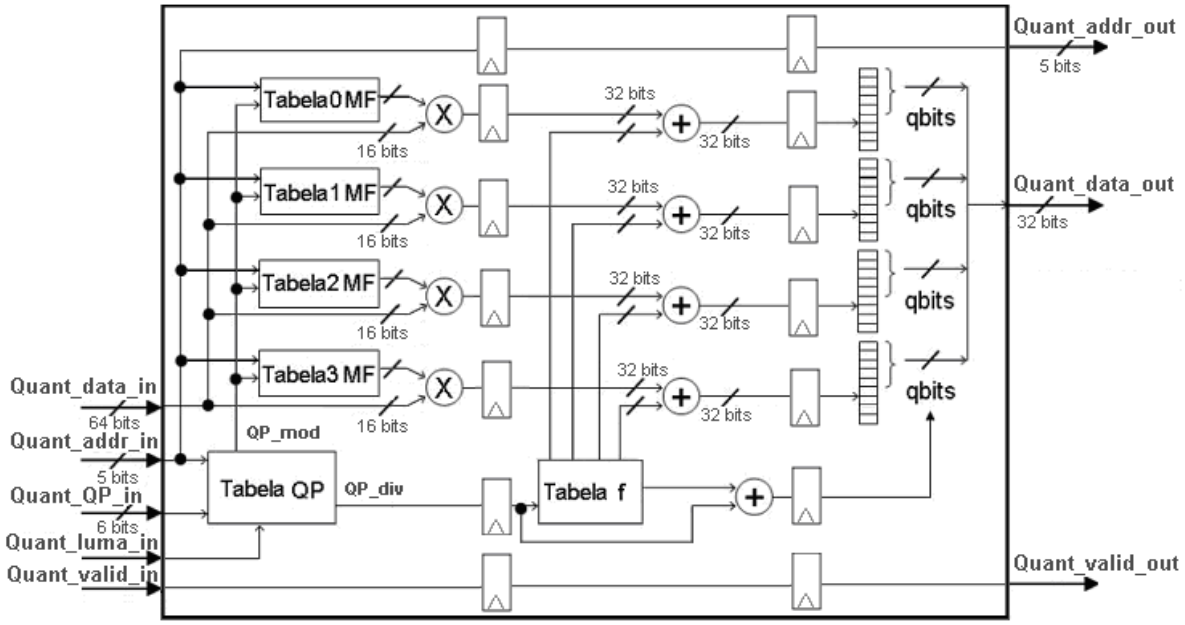


Figura 73: Estrutura interna do módulo de Quantização

Cada um dos três estágios propostos é responsável por uma funcionalidade distinta dentro do algoritmo, conforme descrito a seguir:

- Estágio 1 (processamento de QP e multiplicação): responsável por usar a entrada *QP* para calcular o valor de *MF*. Este bloco precisa estar sincronizado com a leitura de dados do quantizador (*Yij*), pois as coordenadas de cada elemento de entrada precisam ser consideradas para a adequada determinação do valor de *MF*. Percebe-se que o cálculo de *MF* é feito a partir de uma busca em tabela (chamada Tabela QP). Os sinais de validade do módulo e de posição dos dados são propagados entre os vários estágios para sincronizar-se com a saída;
- Estágio 2 (processamento aritmético): responsável pela realização das operações de ajuste do fator *f* para arredondamento do valor anteriormente calculado, além de cálculo da quantidade de bits de deslocamento na saída (*qbits*);
- Estágio 3 (deslocamento): estágio final responsável pelo deslocamento de bits do valor gerado na saída do bloco aritmético de um número de bits igual a *qbits*, representando assim uma operação de divisão simplificada.

De forma geral, conforme ilustrado na Figura 73, os principais sinais implementados para essa solução são:

- *Quant_data_in*: linha de valores de entrada ligados na saída da Hadamard . Cada amostra de entrada é representada por 16 bits;
- *Quant_addr_in*: identificador do número da linha que está sendo informado na porta anterior (*Quant_data_in*);
- *Quant_QP_in*: parâmetro de quantização (QP) do macrobloco de entrada;
- *Quant_luma_in*: identificação de tipo de dado (luminância/crominância);
- *Quant_valid_in*: sinal que indica validade de dados na entrada do módulo;
- *Quant_data_out*: linha de valores gerados pelo processamento do módulo.
- *Quant_addr_out*: identificador de qual linha está sendo gerada na saída;
- *Quant_luma_out*: tipo de dado na saída (luminância/ crominância);
- *Quant_valid_out*: sinal que indica validade de dados na saída do módulo.

5.1.2 Módulo Computacional Inverso

Após a etapa de codificação, os dados devem passar por algoritmos que realizam as operações de reconstrução do vídeo codificado, de forma similar ao que acontece com um decodificador de vídeo. Após passar pelo processo de quantização inversa, deve-se considerar outros três módulos, conforme a seguir descritos:

- Hadamard inversa 4x4;
- Hadamard inversa 2x2;
- DCT inversa 4x4.

5.1.2.1 Módulo de Quantização Inversa

O processo de quantização inversa, por sua vez, é responsável por recuperar a largura de bits original dos dados comprimidos por um codificador de vídeo. Para tanto o algoritmo de quantização inversa deve identificar e multiplicar os dados recebidos pelos mesmos valores individuais usados no processo de quantização direta.

A expressão formal das operações a serem realizadas por este módulo é apresentada a seguir:

$$| W'_{ij} | = | Z_{ij} | . VF_{(0,0)} . 2^{\text{floor}}(QP/6) - 2 \quad (14a)$$

$$\text{sign}\left(W'_{ij} \right) = \text{sign}\left(Z_{ij} \right) \quad (14b)$$

Onde:

W'$_{ij}$ é o elemento de saída da quantização inversa;
Z$_{ij}$ é o elemento de entrada (resultado da operação de quantização);
QP é o parâmetro de quantização;
VF é o fator de escala que depende de *QP* e da posição do elemento na matriz;
Sign() representa o sinal da amostra.

O valor de VF (inVerse Factor), de acordo com a norma H.264, é definido considerando as informações de QP_mod (QP%6), QP_div (QP/6) e a posição (i, j) específica do elemento dentro da matriz de entrada, a partir da Tabela 7.

Tabela 7 – Determinação do valor numérico do fator de escala do quantizador inverso

QP	Posições (0,0) (2,0) (2,2) (0,2)	Posições (1,1) (1,3) (3,1) (3,3)	Outras Posições
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Fonte: Richardson, 2003

Em uma abordagem de hardware a determinação de VF é normalmente implementada por tabelas em memória. O restante das operações necessárias são multiplicações, somas e deslocamento de bits. Tendo em vista a largura de barramento de dados e a vazão do quantizador direto este módulo também adota uma implementação paralela, que permite em sua versão de 32 bits, o processamento de quatro amostras simultâneas (Figura 74).

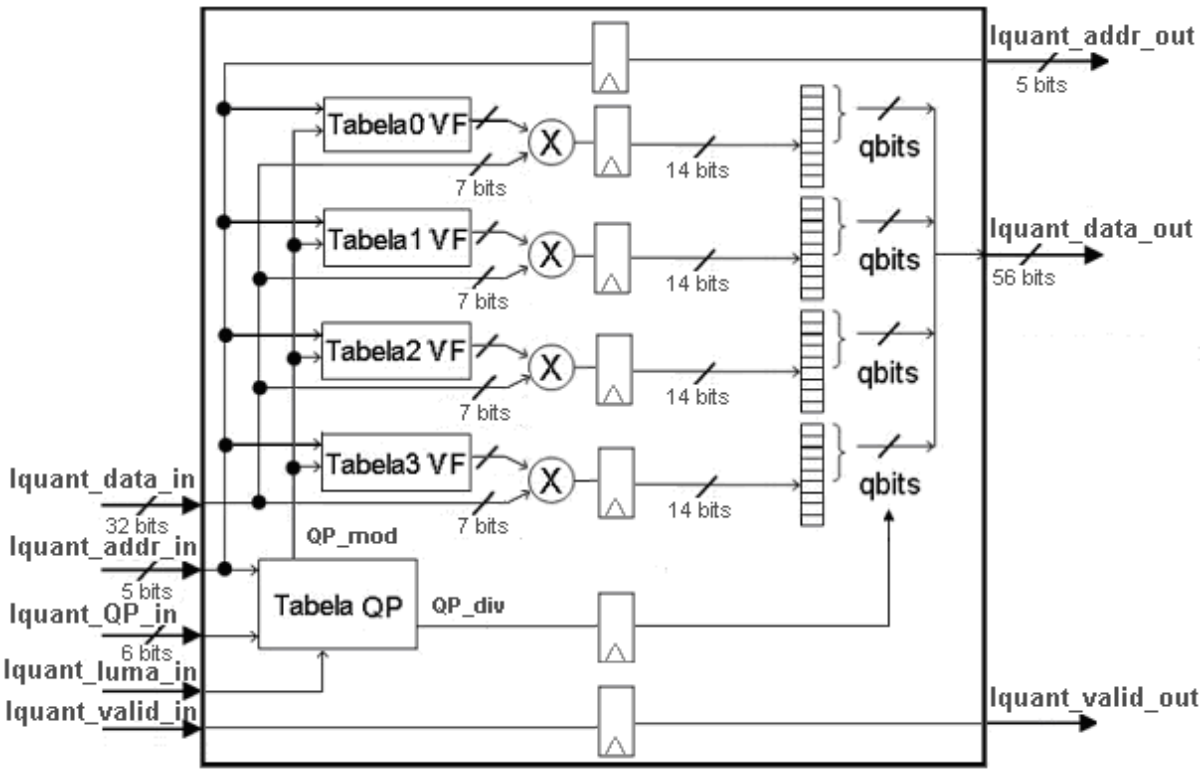


Figura 74: Estrutura interna do módulo de Quantização inversa

Como pode-se observar, a solução adotada opera em um pipeline de dois estágios, onde cada estágio realiza uma operação distinta dentro do algoritmo. A fim de ilustrar isso, a seguir se apresenta a descrição das funções destes dois estágios.

- Estágio 1 (multiplicação): responsável por utilizar o parâmetro de entrada QP para calcular o valor de VF a partir de uma operação de módulo (QP_mod), implementada, na prática, através de acessos a uma tabela (Tabela QP) . Este módulo precisa considerar as coordenadas de cada elemento de entrada na determinação de V*F*.
- Estágio 2 (deslocamento): estágio final responsável pelo deslocamento do valor gerado na saída do bloco aritmético, , implementando, desta forma, uma divisão em hardware.

5.1.2.2 Transformada Inversa Hadamard 4x4

A exemplo de sua versão direta, a transformada Hadamard inversa opera apenas sobre os componentes DC resultantes dos blocos de 4x4 elementos de entrada. Assim, a cada macrobloco avaliado (conjunto de 16x16 elementos), tem-se uma matriz de 4x4 coeficientes DC (caso de amostras de luminância).

Uma característica singular das transformadas Hadamard é que sua operação direta e inversa utiliza as mesmas matrizes de coeficientes H e H^T, conforme figura a seguir:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

Figura 75: Matriz de coeficientes usada para transformada inversa Hadamard 4x4

Assim sendo, sua implementação é bastante similar à arquitetura adotada para sua solução direta, ou seja, sendo composta por dois módulos de transformada unidimensional (IHAD-1D H e IHAD-1D V respectivamente, a fim de montar sua solução bidimensional completa (IHAD-2D). A principal diferença reside no fato da definição do número de bits de cada amostra ser diferente nas versões diretas e inversas.

5.1.2.3 Transformada Inversa Hadamard 2x2

Quando se trabalha com amostras de crominância (Cb e Cr), cujos macroblocos tem dimensão de 8x8 pixels (Figura 67), a etapa de extração dos componentes DC, gera blocos de 2x2 pixels.

Também, neste caso, as matrizes de coeficientes de transformada H e H^T são iguais e definidas por:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figura 76: Matriz de coeficientes da transformada inversa Hadamard 2x2

Devido à grande similaridade, a solução de projeto adotada segue a mesma metodologia da transformada direta 4x4, utilizando assim um *pipeline* de dois estágios apenas, que foi inserido dentro dos módulos anteriormente descritos como IHAD-1D H e IHAD-1D V.

A solução desenvolvida é capaz de realizar tanto operações de luminância como crominância (um sinal de entrada é usado para indicar qual tipo de dados deve ser processado).

A arquitetura completa Hadamard, além dos módulos de cálculo de transformada (IHAD-1D H e IHAD-1D-V), incorpora internamente dois buffers de memória do tipo dupla-porta.

O primeiro buffer, chamado de buffer MB, é usado para armazenar os coeficientes originais, conforme aparecem na entrada.

Já o segundo buffer, chamado de buffer IHAD é responsável por manter os coeficientes DC processados pelo cálculo de Hadamard inversa.

Um módulo de gerência final, chamado de IHAD Comp, se encarrega de acessar ambos os buffers para montar os vetores de saída, sincronizando os valores DC processados com os ACs originais.

Esta estrutura é apresentada na Figura 77.

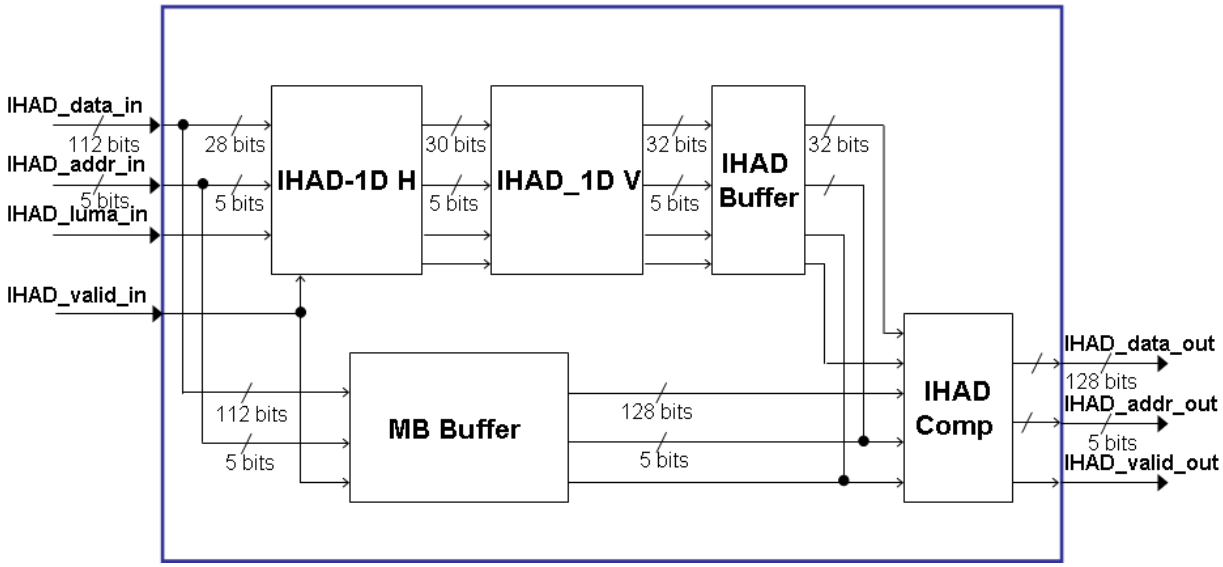


Figura 77: Estrutura interna do módulo IHAD-2D

5.1.2.4 Transformada Inversa DCT 4x4

O algoritmo de transformadas DCT inversas, adotado pelo codificador H.264, funciona de forma bem similar ao algoritmo de transformadas diretas DCT, trabalhando com aritmética de inteiros. Neste caso, as matrizes de transformada inversas A e A^T são:

$$\begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

(a) (b)

Figura 78: Matrizes inteiras da transformada DCT inversa 4x4 (a) direta e (b) transposta.

Pode-se observar que os valores das matrizes variam entre -1/2 e 1/2 em representação inteira, podendo assim ser facilmente implementada por operações de deslocamento de bit para a direita.

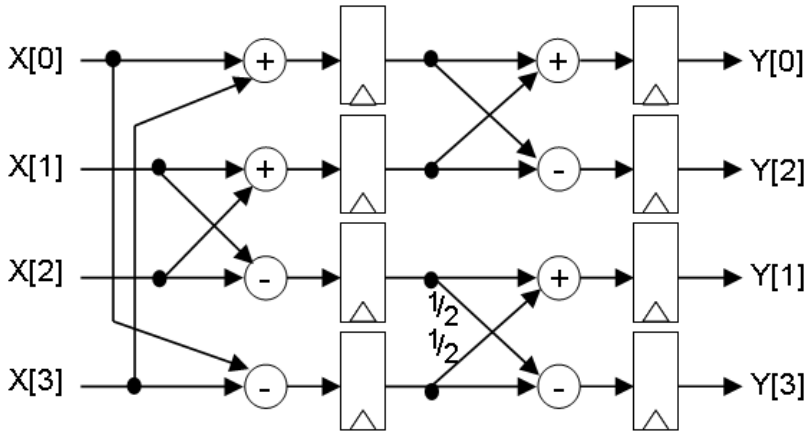


Figura 79: Implementação em hardware de cálculo de uma linha de uma DCT 4x4 inversa

A arquitetura desenvolvida (Figura 79) foi também ajustada para suportar até barramentos de 64 bits, fornecendo simultaneamente vetores de oito pixels. Neste sentido o módulo IDCT-2D foi internamente paralelizado, de forma a operar em dois canais (cada um suportando quatro amostras de um dado). Na prática, são processados simultaneamente duas linhas de blocos vizinhos. A latência total é de oito ciclos de relógio para esta arquitetura duplicada produzindo simetricamente uma concatenação de valores internamente processados de oito amostras reconstruídas a cada ciclo de relógio. Uma representação desta solução é apresentada na Figura 80.

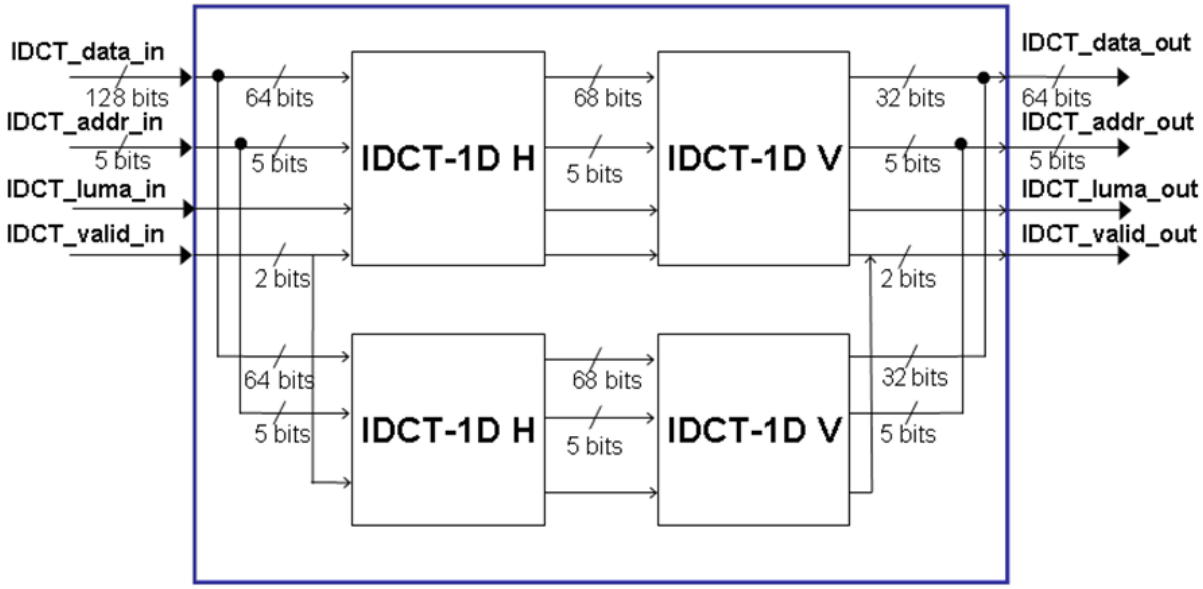


Figura 80: Estrutura interna da arquitetura IDCT-2D duplicada

5.1.3 Integração do Módulo Computacional Intra

A estratégia de projeto adotada considerou características particulares de dependência de dados dentro de um codificador H.264/SVC para compor uma solução realmente eficiente (tão rápida quando as interfaces de comunicação permitiam) através do reuso de hardware e memória.

As soluções desenvolvidas para cálculo de transformadas e quantização buscam uma taxa de dados de saída bastante elevada na saída (ao menos provendo capacidade de processar 16 amostras de vídeo HD).

A idéia é que esta solução possa ser utilizada em uma arquitetura de cálculos recursivos, ou seja uma arquitetura iterativa que permita o reuso de seus módulos internos diversas vezes, para a produção de uma solução multi-camadas. Com isso, a complexidade da solução em hardware não aumenta na mesma proporção do número de camadas.

Uma figura básica que descreve genericamente a arquitetura proposta é apresentada na Figura 81.

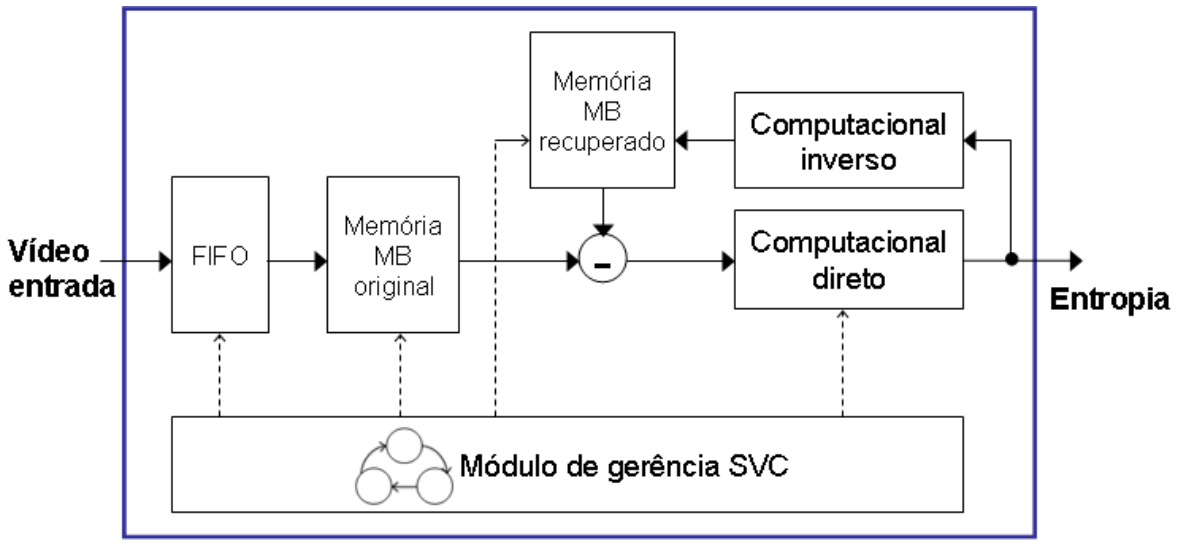


Figura 81: Arquitetura de codificador intra H.264/SVC com abordagem iterativa

Pode-se notar na figura apresentada que a arquitetura desenvolvida permite o reaproveitamento dos blocos computacionais diretos e inversos, a partir do controle realizado por uma máquina de estados principal.

Esta máquina de estados, chamada de módulo de gerência SVC, foi especialmente desenvolvida para controlar e sincronizar os demais módulos. A solução opera com granularidade de macrobloco, ou seja, após receber cada grupo de 16x16 pixels é realizada a codificação da camada base e a seguir as subseqüentes camadas.

De forma geral, a solução desenvolvida integra, além dos módulos computacionais, um módulo de gerência, que controla os fluxos de dados, e uma estrutura de memórias adicionais, responsável por armazenar temporariamente os dados do macrobloco que deve ser processado (original ou recuperado).

5.1.3.1 Módulo de Gerência SVC

O elemento chave para o funcionamento desta arquitetura é o módulo de gerência SVC, que é implementado através de uma máquina de estados finita. Esta máquina coordena todos os módulos da arquitetura de codificação intra, controlando e sincronizando as transferências de dados, ou seja, definindo quando e onde cada memória pode ser lida ou escrita.

Como forma de descrever seu funcionamento interno, foi esboçado, na Figura 82, na forma de fluxograma operativo, o algoritmo suportado pelo módulo de gerência SVC, que foi desenvolvido a fim de implementar a arquitetura desenvolvida de codificação H.264/SVC multi-camadas.

Observando-se a figura, percebe-se que logo após a primeira inicialização, quando informações globais do codificador H.264/SVC são lidas (número de camadas, tipo de dados e valores de QP para cada camada) o módulo de gerência SVC está apto para iniciar a transferência de dados da FIFO de entrada para a memória de MB original.

Cada vez que um novo dado é colocado na entrada do módulo computacional, um bit de validade deve ser gerado para indicar a presença de um dado disponível. Este processo se repete para todos os dados de cada macrobloco.

Na primeira etapa da iteração, os dados desta memória se referem à codificação da camada base.

Assim que os primeiros dados de saída do módulo computacional intra inverso são detectados, a “memória MB recuperado” começa a ser escrita, sendo o contador da camada base incrementado a cada ciclo.

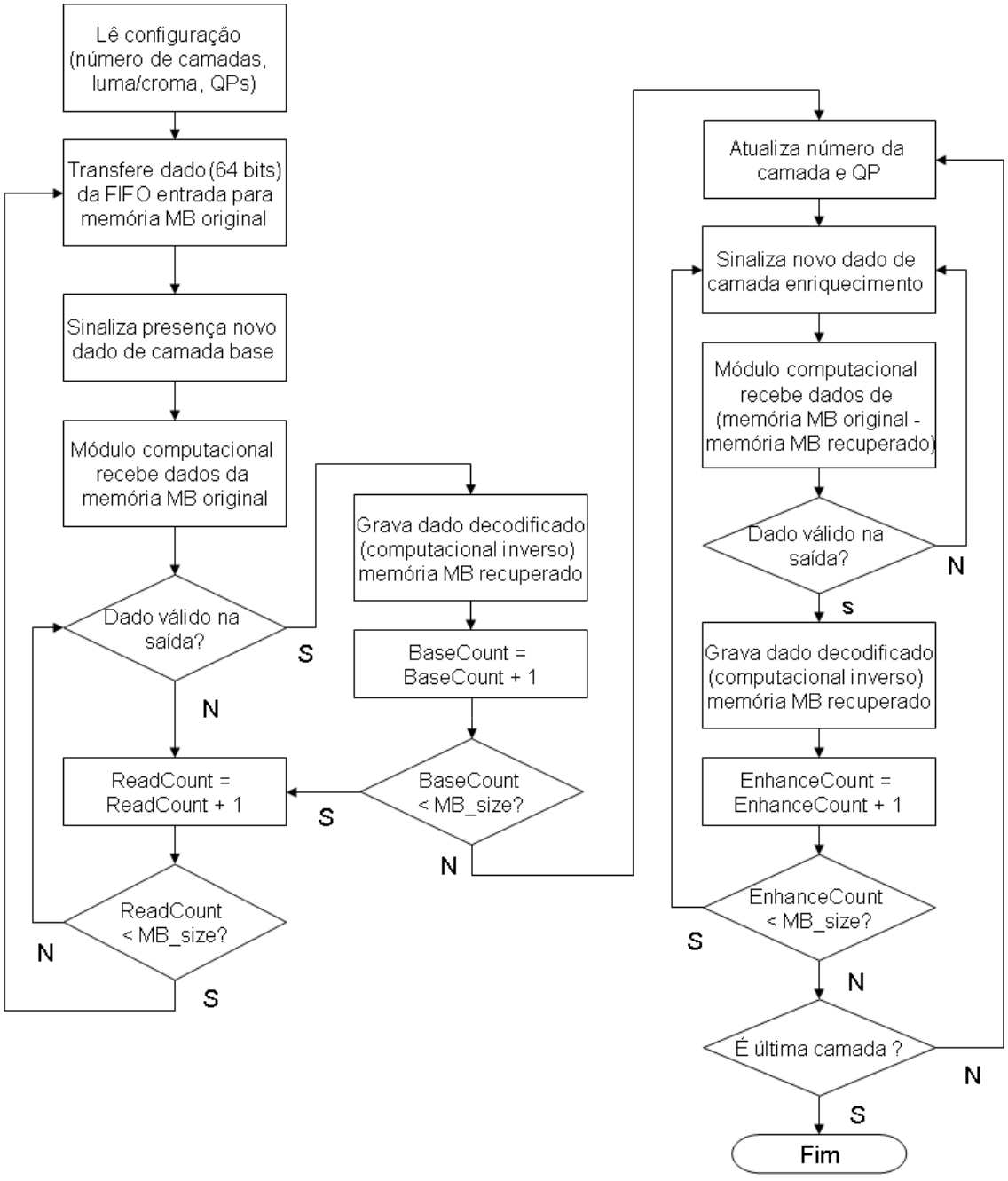


Figura 82: Algoritmo responsável pela codificação multi-camadas iterativa

Quando a quantidade de dados codificados pelo módulo computacional intra atingir o tamanho de um macrobloco completo a máquina de gerência SVC reconhece que um macrobloco da camada base já foi codificado e que a primeira camada de enriquecimento pode ser processada.

Com base nas informações coletadas da transformada DCT inversa, as informações de resíduo são calculadas (subtraindo-se os dados macrobloco original pelo macrobloco recuperado).

Estes dados de resíduo calculados é sinalizado então que novos dados estão disponíveis na entrada do módulo computacional direto, podendo-se se iniciar o processamento da próxima camada.

A cada nova iteração o valor de QP precisa ser reavaliado (um valor individual QP é permitido para cada camada).

Quando novos dados são detectados na saída do módulo computacional intra inverso, a “memória MB recuperado” é novamente sobrescrita. Conforme aumenta o número de camadas processadas (e consequentemente o número de estágios de enriquecimento do vídeo) reduz-se cada vez mais a informação residual.

Este processo de refinamento iterativo é repetido até a última camada de enriquecimento.

5.1.3.2 Memórias de Macrobloco Original e Recuperado

A fim de suportar o processo iterativo descrito, memórias FIFO dupla porta são usadas como interface para os módulos externos e mesmo para garantir o sincronismo das operações internas.

Além disso, memórias dupla porta adicionais são necessárias para armazenar informações temporárias durante cada operação iterativa.

Visando auxiliar o entendimento da estrutura de ligação das memórias implementada por este módulo, as memórias utilizadas são apresentadas, na forma de diagrama de blocos na Figura 83.

A “memória MB original” é responsável por armazenar os dados de entrada de macrobloco do vídeo original, enquanto que a “memória MB recuperado” armazena dados que são gerados a cada iteração.

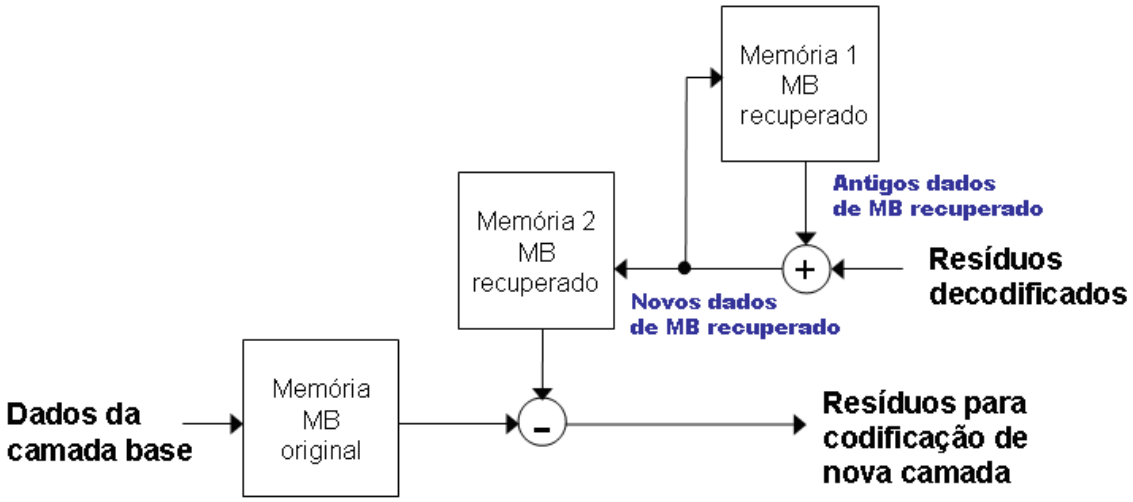


Figura 83: Arquitetura de memórias para utilização em arquitetura iterativa

Na prática, a memória de macrobloco recuperado é composta por duas memórias distintas, uma usada para implementar a operação com os resíduos gerados pela decodificação e outra para ser usada na entrada do módulo de codificação.

Mais particularmente, uma das memórias é lida para se obter o último dado reconstruído, que é, a seguir, somado com o novo resíduo processado e escrito novamente na memória.

Em paralelo este mesmo valor é escrito em uma segunda memória, que permanece sendo continuamente lida pela máquina de controle para gerar a próxima camada sem que, com isso, se cause interrupção no processo de leitura.

Foi utilizada esta estrutura de ligações para se garantir cada memória sofresse no máximo uma leitura e uma escrita por ciclo de relógio. No geral, como as memórias têm tamanho de apenas um macrobloco, o consumo da solução, mesmo com duas memórias, é bastante reduzido.

5.2 MÓDULO DE FILTRO ANTI-BLOCAGEM

O bloco de filtro anti-blocagem, incluso na especificação H.264/SVC é importante para eliminar artefatos decorrentes da digitalização da imagem. O algoritmo proposto pela norma (*deblocking filter*) foi desenvolvido para filtrar grupos de 4x4 pixels quando operando com elementos de luminância ou blocos de 2x2 pixels para crominância, em duas direções: horizontal e vertical (BROTHERTON; BAYARD; HANDS, 2006).

A Figura 84 apresenta o ordenamento tradicional adotado para filtrar um macrobloco de 16x16 pixels.

Inicialmente ocorrem as filtragens horizontais dos blocos de 4x4 pixels mais à esquerda. Nesta etapa, os blocos anteriores (P), destacados em cinza, constituem-se de blocos de um macrobloco vizinho já previamente filtrado (vizinho da esquerda) enquanto que os blocos atuais (Q) são os blocos de 4x4 pixels localizados na esquerda do macrobloco atual. O processo de filtragem ocorre linha a linha de cima para baixo até o final do macrobloco. Durante esta primeira etapa é realizada a filtragem que se refere à borda mais a esquerda do macrobloco, conforme indicado pela seta. A segunda etapa volta a considerar os blocos mais a esquerda do macrobloco, porém, desta vez, sendo considerados como blocos anteriores (P), uma vez que estes foram filtrados, enquanto que os blocos atuais (Q) representam a segunda coluna de blocos 4x4 pixels.

O processo se repete, também da linha superior até a inferior do macrobloco, procedendo a filtragem referente à segunda borda vertical do macrobloco (indicada pela seta). Mais duas etapas de filtragem horizontal são necessárias para completar o macrobloco inteiro. Após isso começa-se a executar a filtragem vertical do macrobloco. A primeira etapa irá considerar como blocos anteriores (P) os blocos 4x4 já filtrados, dos macroblocos vizinhos de cima (indicado em cinza).

As filtragens, neste caso, são realizadas com os pixels alinhados em colunas, ocorrendo no sentido da esquerda para a direita. Após quatro etapas, o macrobloco está inteiramente filtrado, considerando-se o efeito de todos os blocos vizinhos.

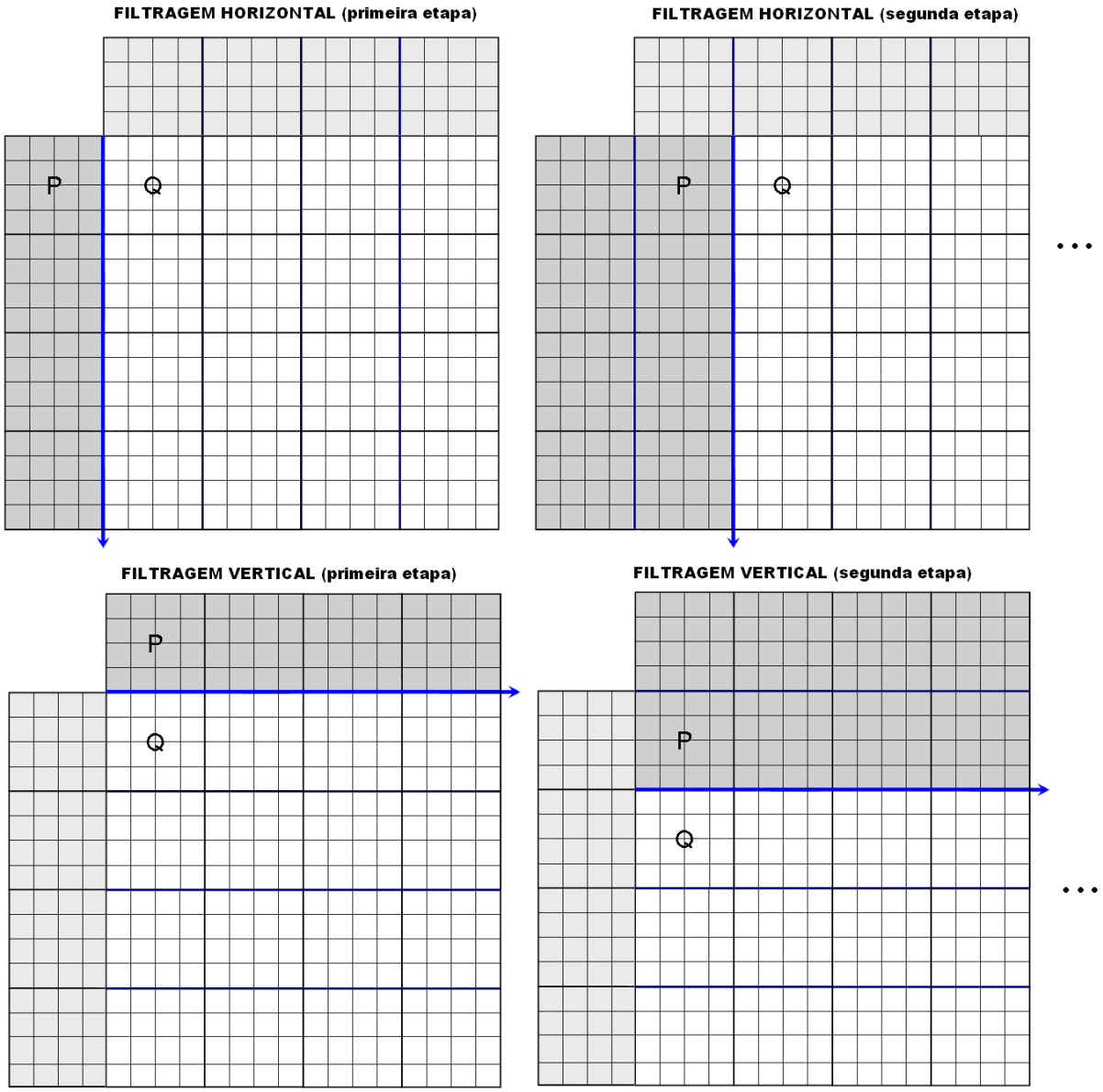


Figura 84: Etapas do filtro anti-blocagem para um macrobloco de 16x16 pixels

Na Figura 85 se apresenta a distribuição de pixels durante uma filtragem horizontal. Deve-se lembrar que, neste algoritmo, dois LOPs são processados a cada vez, correspondendo à filtragem de uma linha isolada.

Neste contexto, a sequência composta pelos pixels $p3$, $p2$, $p1$ e $p0$ corresponde ao bloco anterior (bloco P), enquanto que os pixels $q0$, $q1$, $q2$ e $q3$ correspondem ao bloco atual (bloco Q).

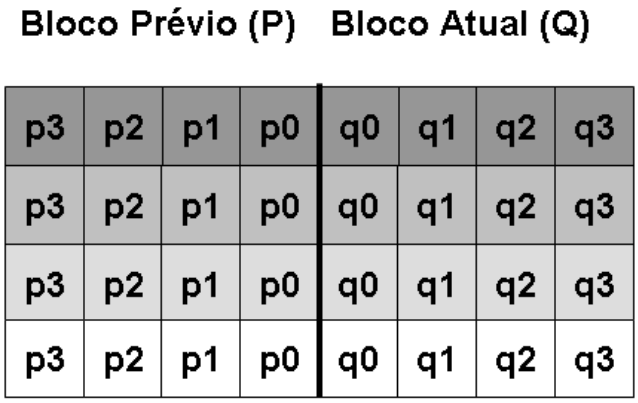


Figura 85: Divisão do bloco em linhas de pixels para filtragem horizontal

A localização das bordas deve ser observada pelo algoritmo, pois, dependendo de onde se encontra a borda, a intensidade da filtragem deve mudar. O parâmetro que determinada esta intensidade vinculada à localização do bloco é chamado de BS (*Boundary Strength*).

Tabela 8 – Determinação do valor de BS de acordo com a situação do bloco

Condição do bloco	BS
Bloco com borda externa (separa blocos de macroblocos vizinhos)	4
Bloco com borda interna (separa blocos de um mesmo macrobloco)	3
Bloco contém informações de resíduos de compensação de movimento*	2
Bloco contém informações de vetores de movimento*	1
Outros casos	0

*Vinculado ao procedimento de estimativa de movimento entre quadros

O valor BS = 0 elimina a realização de filtragem. Os demais são usados pelo filtro como parâmetro de ajuste dos cálculos. O valor BS=4 é considerado como a condição de filtragem mais severa. Na prática, um filtro anti-blocagem possui dois algoritmos distintos: um para BS=4, utilizando parâmetros fixos para máxima filtragem, e outro para $0 < BS < 4$, onde o valor de BS é considerado na determinação de parâmetros internos do filtro.

De forma geral, o filtro anti-blocagem deve atuar primordialmente sobre os pixels mais próximos às bordas (p_0, q_0, p_1 e q_1). Para sua realização deve-se, entretanto, observar o distanciamento presente entre valores de pixels vizinhos.

Neste quesito, são considerados dois parâmetros importantes: *beta*, que determina a máxima diferença entre valores de um mesmo LOP (ou COP) e *alfa* que determina a máxima diferença entre valores consecutivos de LOPs (ou COPs) vizinhos.

Em via de regra, o filtro anti-blocagem só será executado sobre os valores de $p0$ e $q0$ se as seguintes condições forem satisfeitas:

$$| p0 - p1 | < alfa \quad (15a)$$

$$| p1 - p0 | < beta \quad (15b)$$

$$| q1 - q0 | < beta \quad (15c)$$

Da mesma forma o filtro anti-blocagem só será executada sobre os valores de $p1$ e $q1$ se as seguintes condições forem satisfeitas:

$$| p2 - p0 | < beta \quad (16a)$$

$$| q2 - q0 | < beta \quad (16b)$$

A Figura 86 ilustra visualmente as diferenças consideradas para avaliação da aplicação ou não do filtro:

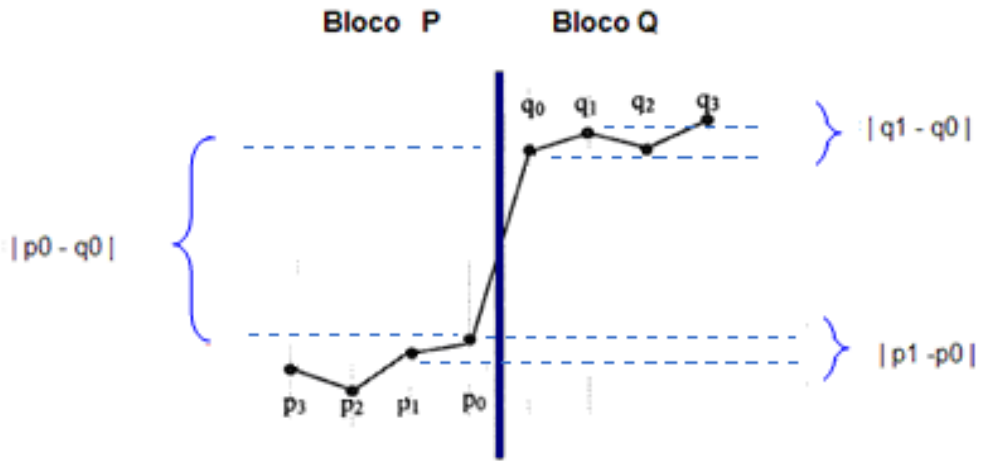


Figura 86: Distâncias entre pixels consideradas no filtro

Os valores dos parâmetros *alfa* e *beta*,que são utilizados para estas decisões, são obtidos a partir de uma tabela fornecida pela norma H.264/SVC , a qual é apresentada na Tabela 9.

Tabela 9 – Tabela para determinação dos valores de *alfa* e *beta*

	indexA																								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

	indexA																								
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α'	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β'	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

Fonte: Richardson, 2003

Como pode-se observar pela análise da Tabela 9, a obtenção dos valores de *alfa* e *beta* do filtro está vinculada a parâmetros denominados *indexA* e *indexB*, os quais, por sua vez, dependem do valor do parâmetro de quantização adotado (QP) e dos valores de *offsetA* e *offsetB*, que são parâmetros estes definidos pelo codificador durante as etapas de compressão de vídeo.

As equações definidas para o cálculo dos parâmetros *indexA* e *indexB* são respectivamente:

$$indexA = \text{Min} \mid \text{Max} \mid 0, QP + offsetA \mid , 51 \mid \quad (17a)$$

$$indexB = \text{Min} \mid \text{Max} \mid 0, QP + offsetB \mid , 51 \mid \quad (17b)$$

Conforme comentado anteriormente, o filtro anti-blocagem, definido pela especificação H.264, possui dois algoritmos independentes: um algoritmo para BS= 4 e outro para $0 < BS < 4$.

A fim de facilitar a compreensão do filtro anti-blocagem completo, os dois algoritmos são, a seguir, apresentados separadamente, indicando-se a sequência de passos que deve ser realizada internamente sobre cada linha (ou coluna) de pixels para a obtenção dos dados filtrados.

Algoritmo para 0< BS< 4

Para o cálculo dos novos valores de $p0$ e $q0$, inicialmente deve-se calcular o valor de diferença, *Dif*, que é obtido a partir da ponderação dos pixels mais próximos da borda:

$$Dif = Clip(-tc, tc, ((((q0 - p0) << 2) + (p1 - q1) + 4) >> 3)) \quad (18)$$

Este valor é obtido a partir da função *Clip* que serve para limitar o valor de saída no intervalo (*-tc*, *tc*), sendo o parâmetro tc derivado de uma tabela também fornecida pela norma (*Clip table*), a qual é obtida a partir dos parâmetros *indexA* e BS já definidos. Esta Tabela 10 representa a *Clip table* com valores de saída dependentes de *indexA* e BS:

Tabela 10 – Tabela para determinação do valor de tc

	indexA																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
bS = 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
bS = 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
bS = 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

	indexA																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
bS = 1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
bS = 2	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
bS = 3	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25

A partir do valor obtido para *Dif*, os novos valores de $p0$ e $q0$ (chamados de $p0'$ e $q0'$) são dados respectivamente por:

$$p0' = Clip(p0 + Dif) \quad (19a)$$

$$q0' = Clip(q0 - Dif) \quad (19b)$$

O cálculo dos novos valores de $p1$ e $q1$ ocorre de forma similar. Inicialmente se calcula o valor de *Dif*, que deve então ser somado aos valores originais. Assim sendo os novos valores de $p1$ e $q1$ (chamados $p1'$ e $q1'$) são respectivamente dados por:

$$Dif = Clip(-tc0, tc0, (p2 + ((p0 + q0 + 1) >> 1) - (p1 << 1)) >> 1) \quad (20a)$$

$$p1' = p1 + Dif \quad (20b)$$

e

$$Dif = Clip(-tc0, tc0, (q2 + ((p0 + q0 + 1) >> 1) - (q1 << 1)) >> 1) \quad (21a)$$

$$q1' = q1 + Dif \quad (21b)$$

Algoritmo para BS = 4

Quando o opera com borda externa (BS=4) a intensidade da filtragem anti-blocagem é mais severa, neste caso atingindo inclusive os pixels mais afastados das bordas ($p2$ e $q2$).

Sua implementação apesar de mais onerosa computacionalmente exige menor dependência de valores. A seguir as expressões para cálculo dos novos valores são apresentadas, inicialmente considerando-se o bloco corrente (bloco Q) e anterior (bloco P)

$$q0' = (p1 + 2p0 + 2q0 + 2q1 + q2 + 4) >>3 \quad (22a)$$

$$q1' = (p0 + q0 + q1 + q2 + 2) >>2 \quad (22b)$$

$$q2' = (2q3 + 3q2 + q1 + q0 + p0 + 4) >>3 \quad (22c)$$

$$p0' = (q1 + 2q0 + 2p0 + 2p1 + p2 + 4) >>3 \quad (22d)$$

$$p1' = (q0 + p0 + p1 + p2 + 2) >>2 \quad (22e)$$

$$p2' = (2p3 + 3p2 + p1 + p0 + q0 + 4) >> 3 \quad (22f)$$

Se o bloco for de crominância os valores são dados por:

$$q0' = (2q1 + q0 + p1 + 2) >> 2 \quad (23a)$$

$$p0' = (2p1 + p0 + q1 + 2) >> 2 \quad (23b)$$

Como visto, a implementação de um filtro anti-blocagem demanda diversas operações vinculadas ao tipo de bloco, relação com bloco anterior e a testes de magnitude entre amostras vizinhas. Na prática, porém, seu desenvolvimento em hardware não é tão complexo uma vez que pode ser implementado através de operadores simples como somadores, deslocadores e tabelas em memória (MIN; CHONG, 2007). De uma forma geral, o algoritmo de um filtro anti-blocagem pode ser montado pelo uso de filtros unidimensionais, um para filtrar na direção horizontal e outro na vertical. Neste contexto a solução se assemelha bastante ao algoritmo sugerido para o cálculo de transformadas, ou seja, os dados resultantes da filtragem horizontal passam por um buffer de transposição, que rotaciona a matriz de pixels, permitindo assim a repetição do filtro anterior, só que neste segundo momento gerando um efeito de filtragem vertical. Os dados resultantes da segunda filtragem devem então ser novamente transpostos antes de serem fornecidos para a saída.

Uma representação simplificada deste mecanismo é apresentada na Figura 87:

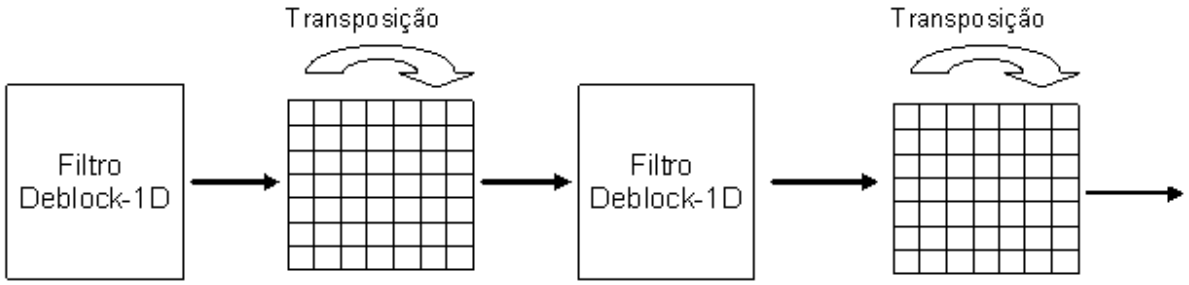


Figura 87: Mecanismo simplificado para implementar filtro anti-blocagem em hardware

Pode-se destacar o trabalho de Li, Goto e Ikenaga (2005), onde um filtro unidimensional recebe dados de dois multiplexadores. Cada multiplexador chaveia dados de uma memória RAM dupla porta ou de um módulo de transposição. Um módulo de transposição adicional é usado para retornar os dados finais filtrados na mesma orientação da memória externa.

Shih, Cheng-Ru e Youn-Long (2006) também utilizam esta técnica, propondo um filtro anti-blocagem baseado em uma arquitetura de pipeline de cinco estágios. Esta abordagem utiliza três módulos de memória local em uma estratégia de reuso de dados.

Liu et al. (2007) apresenta uma proposta similar, mas aprimora o filtro unidimensional para suportar internamente a manipulação de dados nas direções horizontal ou vertical, conforme o tipo de filtragem a ser desempenhada. A área ocupada para a filtragem unidimensional aumenta, mas os módulos de transposição são eliminados.

Chen e Chen (2007) propõem uma arquitetura de filtro rápido por implementar em paralelo ambos os algoritmos anti-blocagem (um para BS=4 e outro para $0 < BS < 4$). Na etapa final um multiplexador é usado para selecionar os dados corretos a serem atualizados.

Rosa (2009) propõe uma arquitetura rápida de filtro anti-blocagem que usa um longo *pipeline* de 16 estágios para implementação desta filtragem. O extenso *pipeline* aumenta a complexidade da implementação, mas reduz o período de trabalho global da solução. Módulos de transposição são usados para alimentar cada estágio de filtragem unidimensional.

Wang et al (2010), por sua vez, propõe a adoção da filtragem vertical antes da horizontal, simplificando assim a lógica de interface com a memória. Além disso, adota um módulo de filtragem mais compacto, baseado em um *pipeline* de três estágios, para realizar a filtragem em cada direção.

A análise destes trabalhou ajudou na definição no desenvolvimento da arquitetura de filtragem em questão. Deve-se, porém, e acima de tudo, se buscar uma solução de filtragem que atenda a demandas práticas, tais como de largura de barramento e taxa de dados oriundas do módulo computacional apresentado anteriormente (transformadas e quantização), evitando a inserção de atrasos adicionais ao sistema, que poderiam reduzir o desempenho global da solução.

Neste sentido, a versão básica deste módulo de filtragem desenvolvido foi projetada para suportar um barramento de 32 bits, operando com uma vazão de quatro amostras por ciclo de relógio. Além disso, foi desenvolvida uma segunda versão, projetada para suportar barramentos de 64 bits (vazão de oito amostras por ciclo de relógio).

5.2.1 Módulo de Filtragem Básico

A exemplo de outros trabalhos, a solução desenvolvida também buscou o aproveitamento de um único módulo de filtragem unidimensional (chamado de Filtro-1D), para realizar, em certo momento, a filtragem horizontal e em outro a filtragem vertical. Na prática a reutilização do mesmo filtro para duas operações não chega a representar um gargalo direto para o sistema proposto, pois esta filtragem ocorre no estágio final de armazenamento do vídeo recuperado, não ficando, portanto, no caminho crítico do laço de retorno (cálculo dos dados codificados e preenchimento da FIFO de saída).

A solução desenvolvida faz uso de um processo seqüencial composto por quatro módulos (Buffer PQ, Filtro-1D, Transp e RAM DP) mais um módulo de controle (Figura 88).

O módulo buffer PQ é composto por duas memórias de dupla porta, que são responsáveis por manter atualizados os valores de blocos P e Q da entrada do módulo de filtragem. É importante destacar que a entrada do módulo utiliza um barramento de 32 bits, enquanto que o módulo de filtragem, para poder operar, internamente, precisa de dois conjuntos de 32 bits, um referente a cada bloco (P e Q).

Sendo assim no início do processo de filtragem das bordas da imagem, o buffer só poderá passar valores para o módulo de filtragem após receber dois conjuntos de dados. Daí em diante, os novos blocos vão sendo montados sistematicamente, considerando os novos dados recebidos externamente (blocos Q) em conjunto com dados processados na etapa anterior de filtragem, a fim de implementar o procedimento indicado na Figura 88. Desta

forma, quando o bloco Q de uma etapa de filtragem, é considerado como bloco P na etapa subsequente.

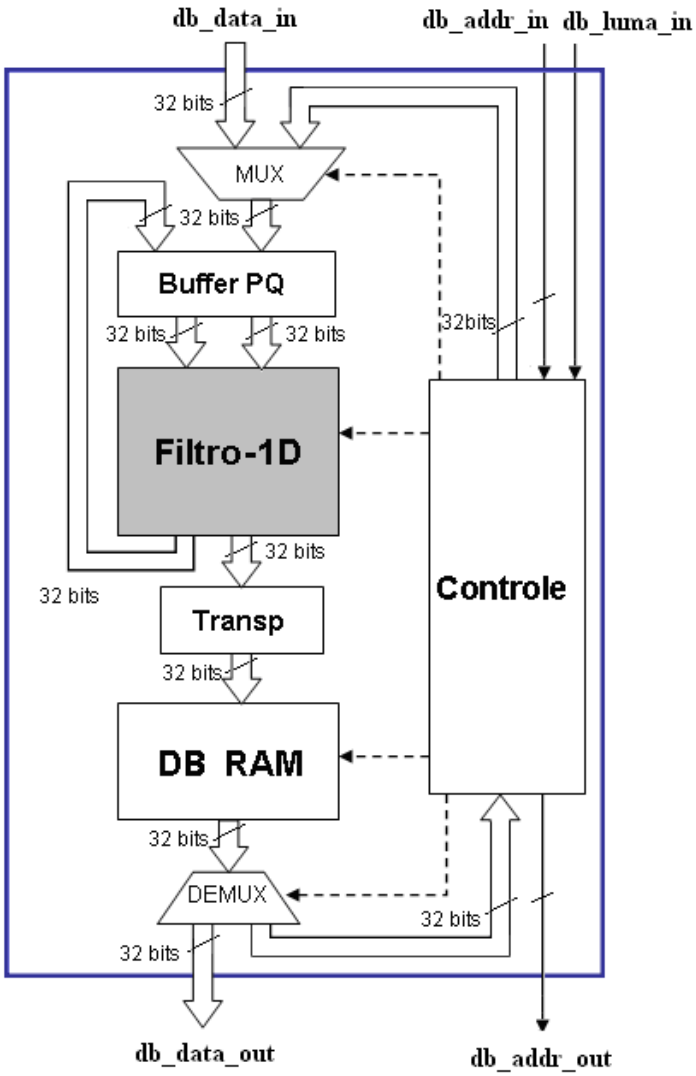


Figura 88: Arquitetura proposta de filtro anti-blocagem básico

O módulo Transp, responsável pela transposição de blocos, é composto por duas matrizes de registradores com dimensão de 4x4 bytes. Cada matriz comporta assim um bloco completo de 4x4 elementos de saída do filtro (bloco Q). Em fluxo de operação cada vez que uma matriz é preenchida, internamente a entrada é chaveada para a segunda matriz, e os dados da primeira são colocados para fora com alinhamento transposto de linha para coluna ou vice-versa.

O módulo DB RAM, indicado na parte inferior da Figura 89, representa um módulo de memórias de dupla porta capaz de armazenar o conjunto de dados filtrados nas últimas operações. Seu tamanho pode variar dependendo da quantidade de macroblocos que se pretenda filtrar a cada sequência de codificação. Para garantir maior independência com a memória externa, porém, pode-se, a priori, definir como tamanho padrão o equivalente a um *slice* completo (uma linha de macroblocos que abrange a mesma largura da imagem) dividido por quatro (ou seja, por utilizar um barramento de 32 bits, cada posição pode armazenar quatro amostras). Alguns autores definem uma memória pequena que comporte apenas um macrobloco, porém para o projeto em questão, onde o filtro pode ser alimentado por rajadas contínuas de vários macroblocos, esta abordagem acabaria exigindo uma FIFO de entrada de tamanho de um *slice*, o que na prática levaria ao consumo da mesma quantidade de memória.

O módulo de controle é uma máquina de estados finita, responsável pela sincronização das transferências de dado (leitura/escrita de memória e interface de entrada/saída) a fim de manter o módulo de filtragem operando no mesmo fluxo de entrada. Além disso, este módulo é responsável por controlar o sequenciamento do processo de filtragem, inicialmente realizado na direção horizontal para a direção vertical. Para tanto o módulo deve fazer uso dos dados previamente calculados e armazenados na memória dupla porta (DB RAM) e utilizá-los como valores de entrada para um novo laço de filtragem. Após terem sido filtrados nas duas direções os dados podem ser disponibilizados para fora do módulo completo.

O último módulo do projeto, Filtro 1-D, é responsável pelo procedimento de filtragem em si. Este módulo foi projetado usando uma arquitetura de *pipeline* de apenas três estágios. Uma solução de *pipeline* tão pequeno somente foi possível pelo fato da solução poder aproveitar-se de informações que transitam junto a cada macrobloco nesta solução (tipo de macrobloco, posição relativa na imagem entre outros), o que facilita as operações de cálculo de parâmetros do filtro, tais como *BS*, *indexA* e *indexB*. Outra técnica utilizada para reduzir a latência da implementação foi realizar o procedimento de avaliação das distâncias absolutas (equações 15a a 15c), que são usadas para determinar a validade do filtro, em paralelo com os cálculos, sendo considerados apenas no último estágio do *pipeline*.

Para facilitar seu entendimento uma representação mais detalhada desta solução é apresentada na Figura 89, onde se pode observar a divisão das tarefas de cálculo de parâmetros e filtragem dentro de uma estrutura em *pipeline* de três estágios.

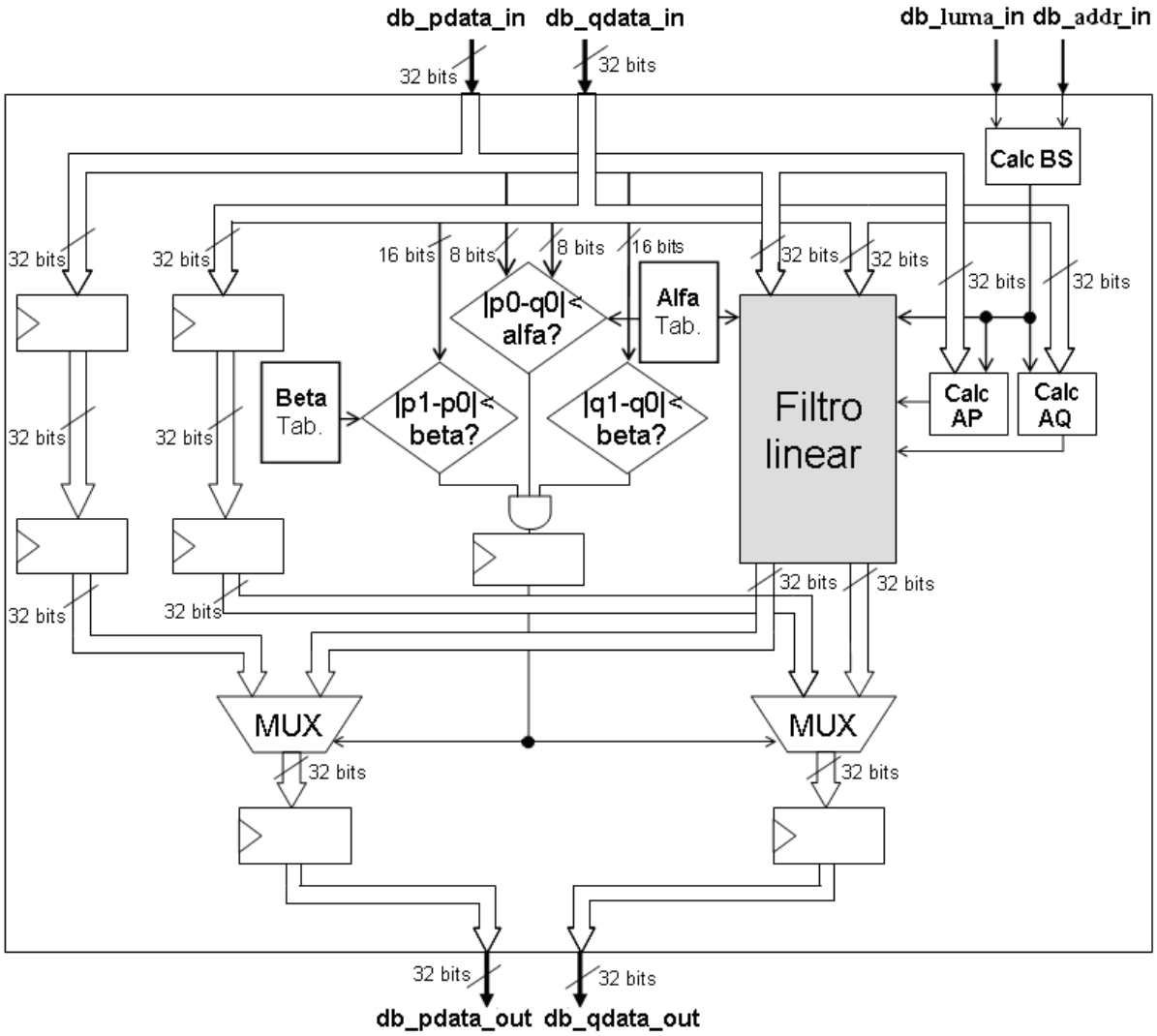
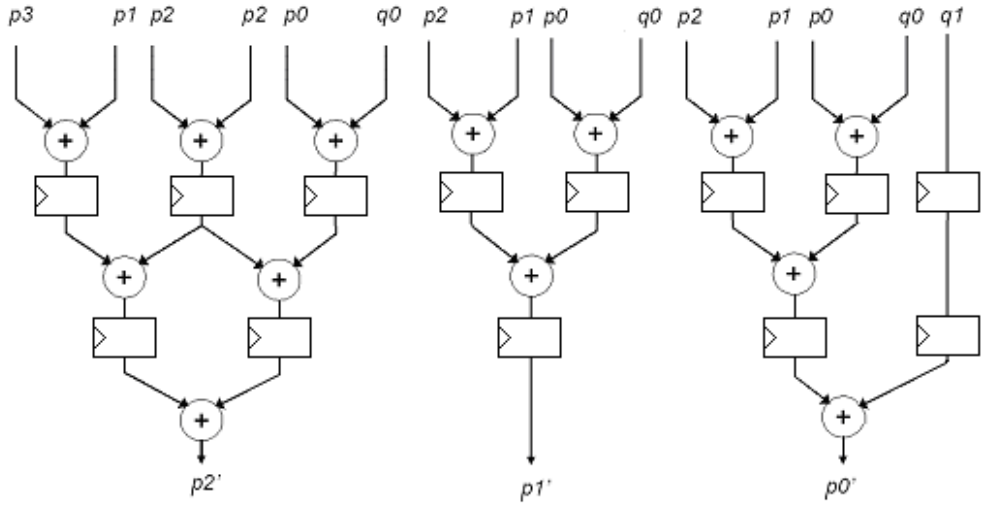


Figura 89: Arquitetura de módulo Filtro-1D básico

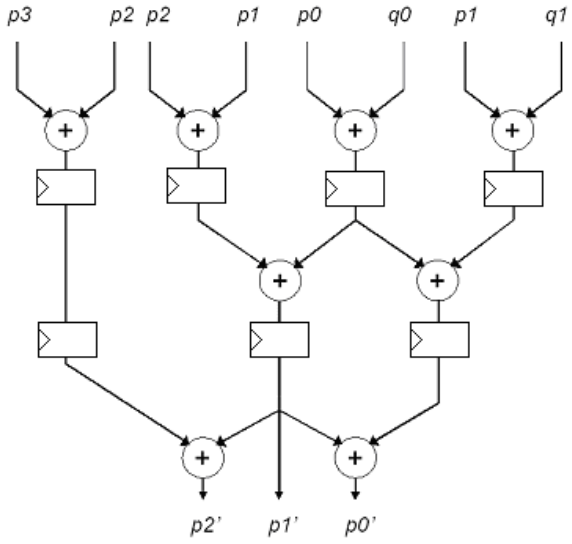
No primeiro estágio são calculados os valores de AP e AQ, que são utilizados nos cálculos de pontos intermediários dos vetores de passado (P) e atual (Q), respectivamente. Na esquerda os dados passam por barreiras de registradores, que atrasam seus valores por dois ciclos de relógio. Assim caso os testes de validade do filtro não sejam verdadeiros os valores de entrada devem apenas ser repassados para a saída sem serem afetados. Esta operação é realizada por multiplexadores de 32 bits na saída do filtro.

O bloco interno referenciado como Filtro linear é composto por uma estrutura de somadores e deslocadores dedicada para fazer os cálculos de filtragem propriamente ditos. A solução, que também adota uma abordagem de *pipeline* de três estágios, reorganiza as operações internas para propiciar o reuso de cálculos parciais por mais de um caminho, reduzindo assim a complexidade da solução sem alterar o desempenho.

Para ilustrar isso, a Figura 90 apresenta o circuito otimizado do filtro anti-blocagem para a condição de BS=4. Após a otimização (Figura 90-b), somente oito somadores e sete registradores são usados, o que representa uma redução da complexidade computacional, principalmente quando comparados com 13 somadores e 13 registradores do algoritmo original.



(a)



(b)

Figura 90: Algoritmo de filtragem para BS=4

(a) antes e (b) após otimização.

Otimizações similares (como parte do cálculo de Dif e procedimentos para blocos de luminância e crominância) foram também implementados nos circuitos para o algoritmo de 0<BS<4, o que causou uma redução global da área total da solução em torno de 37%.

5.2.2 Módulo de Filtragem Estendido

Além da versão anteriormente apresentada (módulo de filtro com entrada em 32 bits) foi desenvolvida uma solução estendida para operar com 64 bits. A princípio a ampliação do barramento de entrada, para receber valores de 64 bits, parece ser resolvida simplesmente com a duplicação das operações internas, porém, na prática, esta ampliação representa um problema mais sério para o módulo de filtragem.

Para entender o problema deve-se considerar que, no contexto da arquitetura proposta, quando trabalhando com barramentos de 64 bits, cada acesso representa um agrupamento de duas linhas de pixels vizinhos, ou seja, duas linhas de blocos geometricamente consecutivos. Uma representação deste alinhamento está apresentada na Figura 91, onde se pode observar que uma mesma informação de 64 bits contém quatro amostras do bloco 0 e quatro amostras do bloco 1 (ou analogamente quatro amostras do bloco 2 e quatro amostras do bloco 3).

O problema que este alinhamento gera está relacionado com a forma com que os dados chegam ao módulo de filtragem. Em um procedimento convencional de filtragem com barramentos de 32 bits, inicialmente deve-se realizar a filtragem horizontal do bloco 1 (Q) em relação ao bloco 0 (P). A seguir se filtra horizontalmente do bloco 2 (Q) em relação ao bloco 1 (P), o mesmo que já havia sido previamente filtrado e alterado pelo estágio anterior. Depois seria a vez bloco 3 (Q) em relação ao bloco 2 (P) e assim por diante.

Quando se trabalha com um barramento de 64 bits, entretanto, o filtro recebe, em um dado momento, os dados referentes aos blocos 0 e 1 e em outro os dados dos blocos 2 e 3. Em nenhum momento são fornecidos os dados do bloco 1 e 2 na entrada do módulo de filtragem.

Se o módulo de filtragem fosse interromper o processo de recepção de dados para calcular esta filtragem intermediária (representada na figura pela borda central em destaque), então o módulo não estaria suportando o processamento com a mesma vazão de entrada. Isso levaria a uma redução do desempenho do módulo de filtragem. Dependendo da taxa efetiva praticada, esta limitação poderia inclusive afetar diretamente o desempenho global do sistema.

Para evitar este problema, a versão anterior do módulo de filtragem foi adaptada para uma arquitetura estendida com um *pipeline* mais longo, capaz de absorver e processar de forma transparente barramentos de 64 bits.

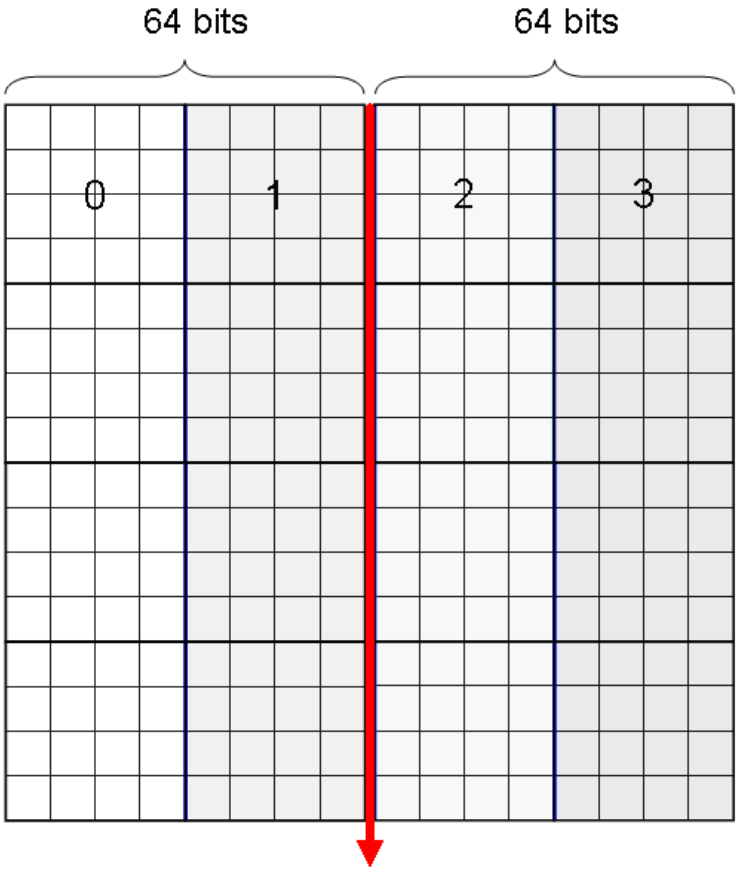


Figura 91: Alinhamento dos pixels na entrada do módulo de filtragem estendido

Uma representação desta arquitetura estendida é apresentada na Figura 92, onde se observa a duplicação dos barramentos internos para acompanhar a mesma largura da entrada.

Nesta nova solução o módulo Buffer PQ foi removido, uma vez que a funcionalidade de alinhar os barramentos de entrada a valores já processados em blocos P e Q passou a ser desempenhada internamente pelo próprio módulo de filtragem unidimensional (Filtro-1D).

Outra mudança importante está na duplicação dos módulos de transposição (Transp), a fim de processar simultaneamente os dois blocos gerados em paralelo pelo módulo de filtragem (saída de 64 bits). Também a memória de dupla porta (DB RAM) teve seu barramento estendido para 64 bits.

Já o módulo de controle em si apresentou pequenas variações (apenas ajuste de alguns contadores e índices de memória), uma vez que, de modo geral, as suas funcionalidades permanecem as mesmas (controle do multiplexador de entrada, ativação do filtro e repasse de dados da memória para a entrada do filtro novamente).

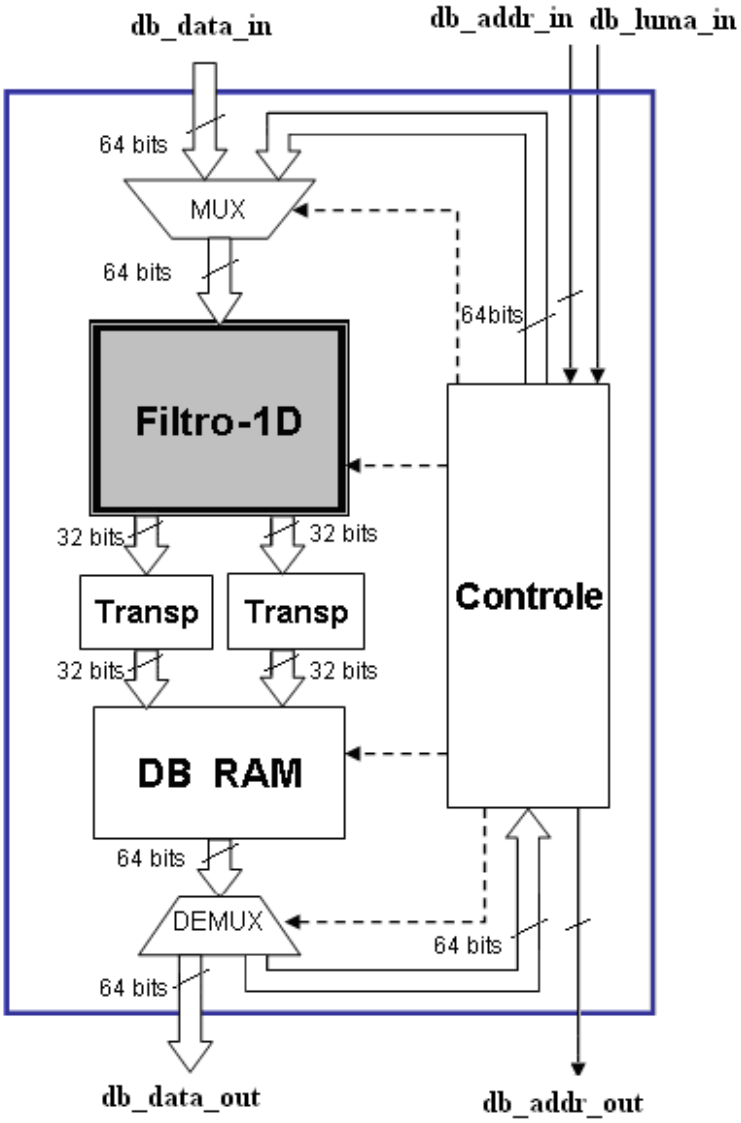


Figura 92: Arquitetura proposta de filtro anti-blocagem estendido

A representação interna do módulo Filtro-1D estendido é apresentada na Figura 93. Como pode-se perceber o módulo adota uma arquitetura de *pipeline* de seis estágios.

No início do primeiro estágio apenas a parcela referente ao bloco menos significativo é utilizada (<p3, p2, p1, p0>). Isso, pois estes dados novos precisam ser inicialmente filtrados considerando os dados do seu bloco anterior, o qual já deve ter sido processado em um procedimento de filtragem prévio.

Na solução adotada, os dados deste bloco anterior já devem estar previamente armazenados em uma memória de dupla porta dedicada, chamada internamente de memória de Bloco P.

As informações desta memória de bloco P anterior são apresentadas na figura com coloração em destaque, sendo seus dados referenciados como grupo <b3, b2, b1, b0>, para evitar conflito com as nomenclaturas das informações de entrada (<p3, p2, p1, p0> e <q3, q2, q1, q0>).

Em uma analogia com o exemplo anterior que citava os blocos 0 e 1 em um conjunto de dados e os blocos 2 e 3 em outro conjunto, esta memória traria armazenada a informação do último bloco 1 filtrado. Assim quando chegam as informações referentes aos blocos 2 e 3, o primeiro procedimento do filtro é resgatar estas informações e, juntamente com os dados do bloco 2 , proceder assim a filtragem correspondentes aos blocos 1 e 2.

O procedimento para realizar esta primeira filtragem leva três ciclos de relógio para ocorrer. Enquanto isso os dados do bloco mais significativo (<q3, q2, q1, q0>) são replicados sem alteração nestes três ciclos de relógio.

Este primeiro procedimento de filtragem irá gerar dois grupos de 32 bits filtrados. O conjunto menos significativo detém os dados do bloco P anteriormente filtrado, enquanto que o conjunto mais significativo contém os dados filtrados do novo bloco P de entrada.

Como o conjunto menos significativo já passou por duas filtragens (uma decorrente de um procedimento anterior e uma segunda realizada nesse estágio), este já pode ser disponibilizado para fora do módulo. Entretanto como o *pipeline* tem profundidade de seis estágios, este sinal precisa ser atrasado por mais três ciclos de relógio para ai sim ser disponibilizado de forma sincronizada na saída.

Já o conjunto mais significativo, chamado de grupo <n3, n2, n1, n0>, precisará passar por outro processo de filtragem, agora considerando seu vizinho da direita (<q3, q2, q1, q0>). Esta nova filtragem levará mais três ciclos de relógio para acontecer, gerando novamente dois conjuntos de 32 bits.

O conjunto menos significativo pode ser disponibilizado para fora do módulo (juntamente com os valores anteriores gerados) enquanto que o mais significativo é salvo na memória de bloco P, para ser usado em outro procedimento de filtragem de 64 bits.

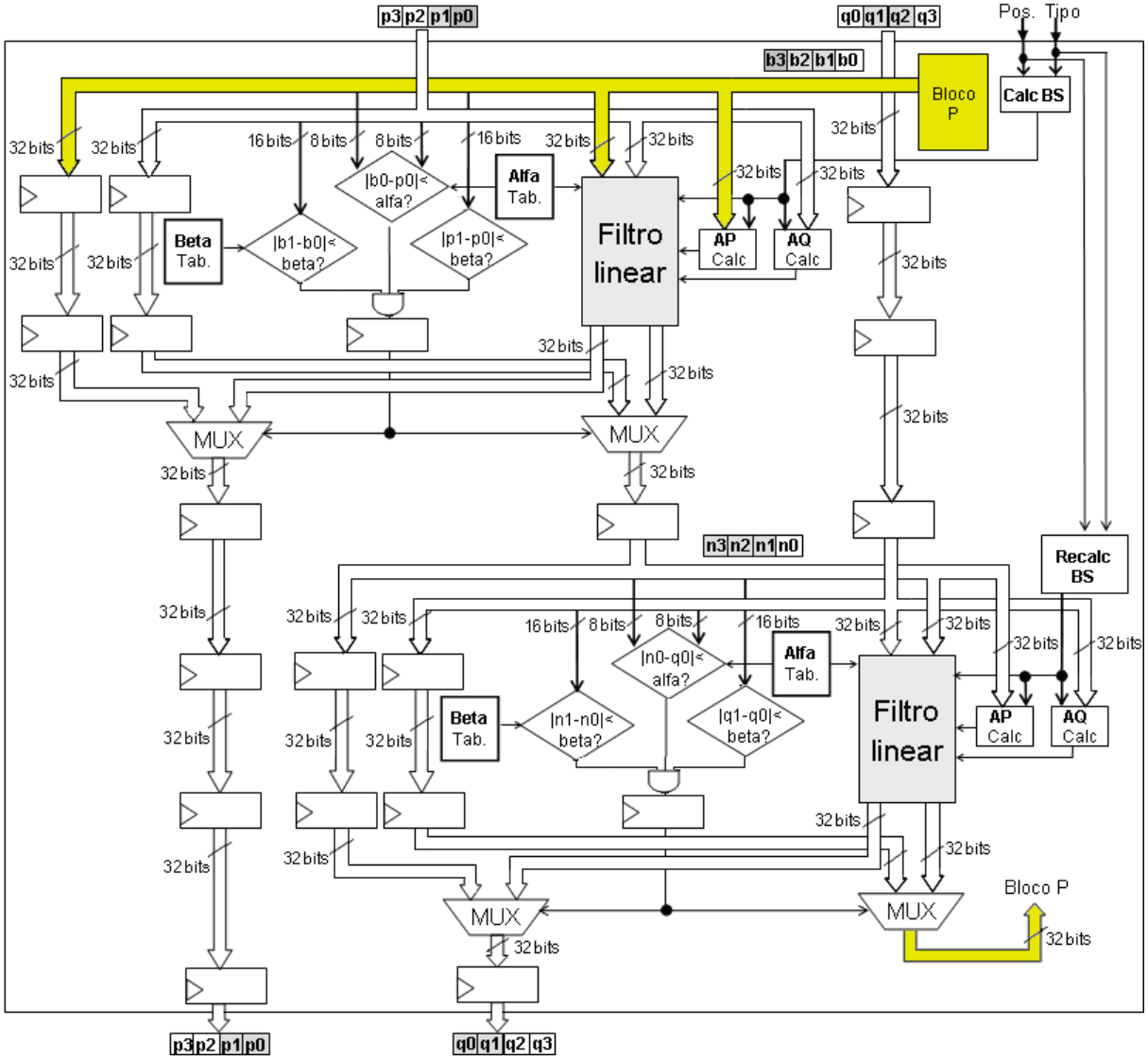


Figura 93: Arquitetura de módulo Filtro-1D estendido

Os módulos de Filtro linear não sofreram alteração na versão entendida de 64 bits, mantendo-se com as mesmas funções e barramentos, ou seja foi utilizada a mesma implementação da versão básica de filtro (32 bits).

5.3 ARQUITETURA PARA CODIFICAÇÃO INTER

5.3.1 Avaliação do Algoritmo de Predição

A principal diferença entre a primeira solução implementada (codificação intra) e a esta segunda (codificação inter) reside na inclusão de um módulo de predição entre camadas, o qual nesta solução também foi implementado em hardware.

O módulo de predição, conforme já descrito, é o bloco responsável por detectar redundância temporal (inter) ou espacial (intra) entre quadros distintos de uma mesma seqüência de vídeo. O mecanismo de busca e comparação é computacionalmente intensivo, visto que é necessário analisar dezenas a centenas de blocos ao redor do ponto atual da imagem, verificando e tentando localizar uma região igual ou muito similar (BRUNIG; NIEHSEN, 2001). Sua função principal é descobrir a distância e as diferenças registradas entre blocos similares do quadro de vídeo atual e de quadros de referência já processados. A informação de distância é determinada como vetor(es) de movimento, que representa(m) valores de deslocamento e direção que relacionam o bloco atual com bloco(s) selecionado(s) no(s) quadro(s) de referência. Isso, pois, na prática, o codificador H.264 pode utilizar múltiplos quadros de referência (XU; CHEN; HE, 2003).

A região onde as pesquisas por similaridade são realizadas é denominada janela de busca (KEITH, 2004). O tamanho da janela de busca é um número que representa a quantidade de pixels que compõem os lados de um quadrado sobre o ponto central da área a ser analisada. Por exemplo, uma janela de busca com dimensão de 96, a partir do ponto central, tem 48 pixels para cada lado (esquerda, direita, cima e embaixo) (KIM et al., 1998).

O módulo de predição tem grande representação na demanda computacional de um codificador de vídeo. Dependendo do algoritmo utilizado sua implementação pode consumir até 80% da área do chip (HUSEMANN, 2006b). O algoritmo que obtém a melhor localização do bloco mais similar na memória de referência é conhecido como algoritmo de busca completa (*full-search*).

Este algoritmo pesquisa na totalidade da janela de busca da(s) imagem(s) de referência por um macrobloco que seja o mais similar possível do bloco da imagem atual. Dessa forma, o procedimento descrito anteriormente é efetuado para todos os possíveis blocos da imagem,

verificando se o macrobloco foi movimentado para qualquer outro ponto da imagem dentro da janela de busca.

Dentre os trabalhos na área pode-se destacar (HE, 2004), que apresenta uma solução tempo-real com resolução de pixel, (KIM; HWANG, CHAE, 2005), que desenvolveu uma arquitetura que trabalha com sete diferentes tamanhos de bloco, (ZANDONAI, 2004), que utiliza uma estratégia de cálculo utilizando múltiplos processadores dedicados operando em paralelo e (NUNO, 2005), que propõe uma arquitetura otimizada utilizando busca hierárquica, decimação de pixels e redução de precisão. Diversas outras soluções são propostas visando-se aumento de paralelismo na estimativa de movimento por busca completa (Figura 94).

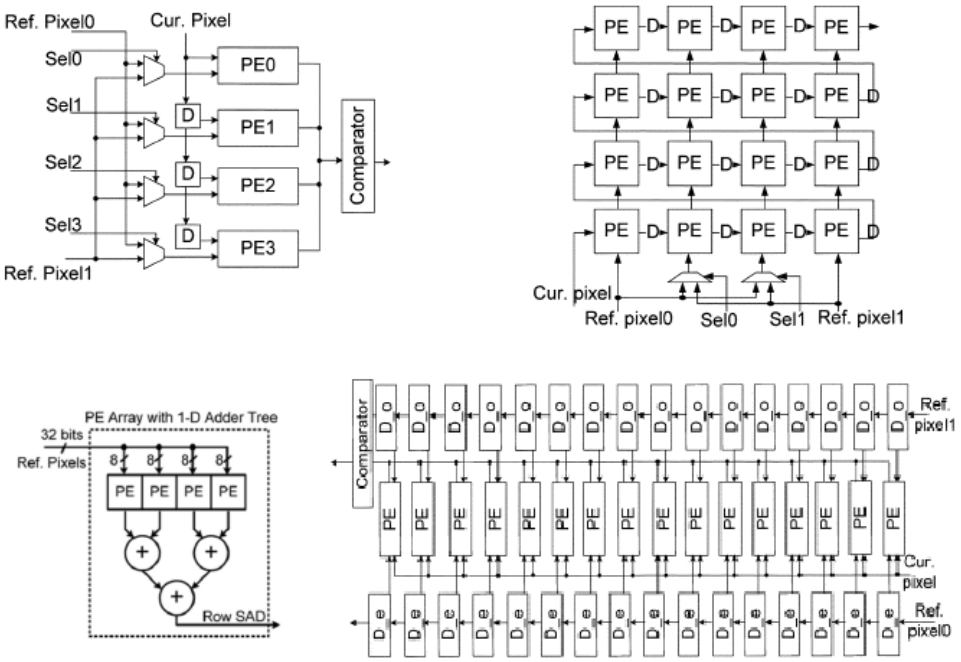


Figura 94: Diversos métodos de estimativa de movimento por busca completa.

Pelo fato de buscar informações em toda a imagem, o algoritmo de busca completa gera um custo computacional bastante elevado devido à quantidade de operações necessárias (YANG; WOLF; VIJAYKRISHNAN, 2005). Além dos algoritmos de busca completa destacam-se os algoritmos de busca esparsa, que procuram evitar a análise de todos os blocos da janela de busca, optando por um padrão de pesquisa, que infere o caminho da movimentação para encontrar o bloco mais parecido. Alguns dos algoritmos de busca esparsa mais conhecidos são o *Three-Step Search* (TSS), que basicamente iniciou os trabalhos nessa linha (KOGA et al., 1981), e também suas evoluções, o *New Three-Step Search* (NTSS) (LI; ZENG; LIOU, 1994) e o *Four Step Search* (FSS) (PO; MA, 1996).

Outra proposta computacionalmente menos intensiva e que leva a menores erros de localização (erro de mínimo local) foi apresentada por Zhu e Ma (2000) e ficou conhecida como pesquisa em diamante (DS - *Diamond Search*).

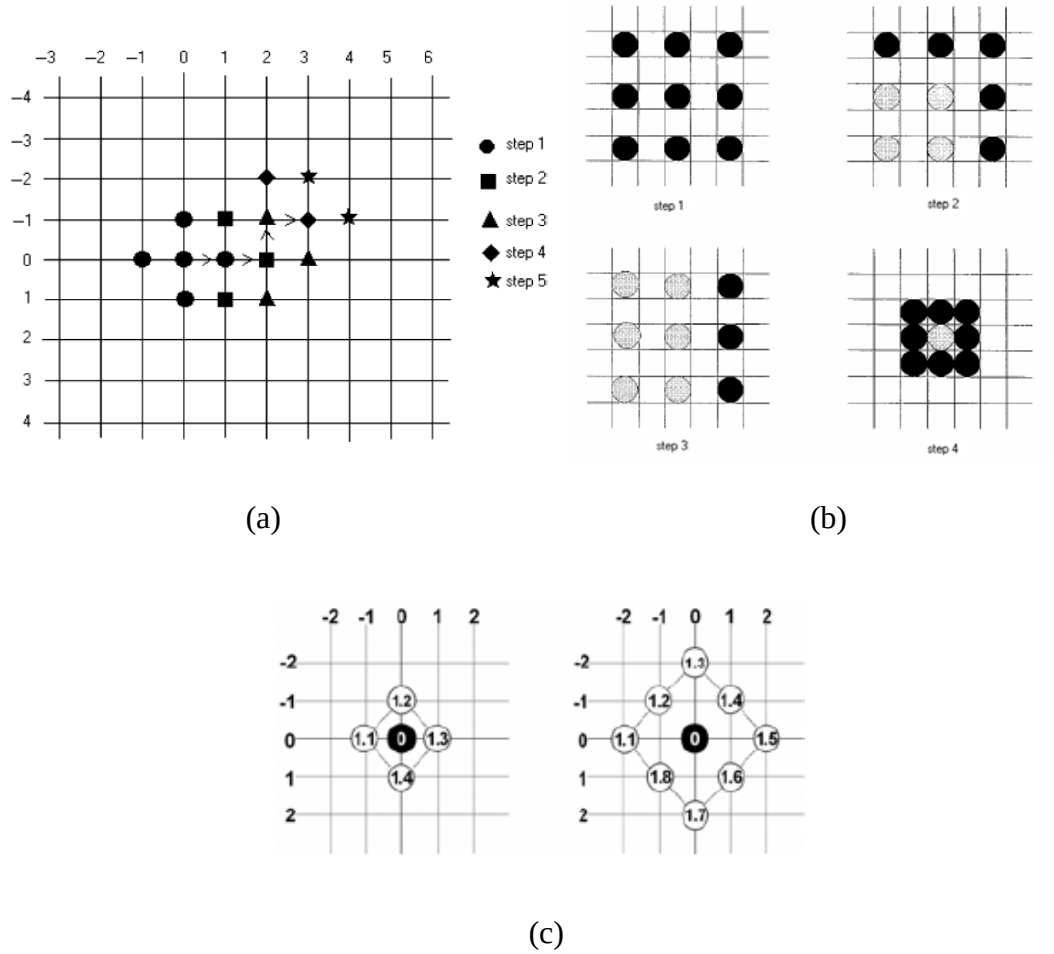


Figura 95: Algoritmos de pesquisa esparsa (a) TSS, (b) FSS e (c) DS.

Em seu trabalho, Porto (2008) faz uma análise sobre diversas configurações práticas de algoritmos de predição de movimento (variação de área de busca, condição de amostragem e refinamento), visando reduzir o tempo de predição. Após diversas análises onde se compararam variações das técnicas de busca completa e pesquisa em diamante, chegou-se ao modelo que usa a técnica DS com uma subamostragem de pixels, seguindo a arquitetura apresentada na Figura 96, onde nove blocos candidatos são calculados, de acordo com o algoritmo DS, em paralelo por unidades de processamento independentes (PU0 a PU8), cujos resultados são enviados a um comparador. Se o resultado final não for obtido na primeira iteração, a unidade de controle atualiza as unidades de processamento para nova comparação.

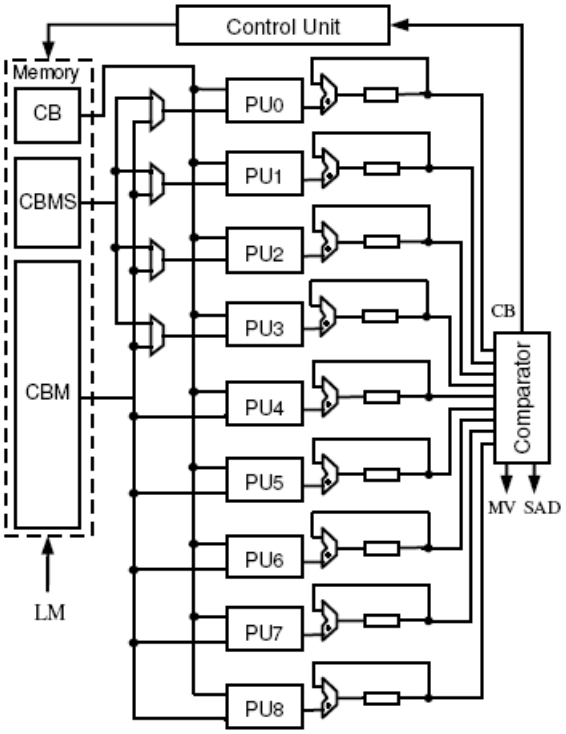


Figura 96: Arquitetura rápida de um módulo de predição DS.
(PORTO, 2008)

Diversas outras estratégias de mecanismo por busca esparsa têm sido apresentados nos últimos anos para agilizar a convergência da estimativa de movimento.

Pode-se destacar nesta categoria o algoritmo *Predictive Motion Vector Field Adaptive Search Technique* (PMVFAST), como um aprimoramento do algoritmo de busca em diamante, que, para reduzir o tempo de busca, toma como ponto de partida para seu procedimento de busca um vetor de movimento previamente obtido através das experiências passadas, em uma estratégia conhecida como busca zonal (TOURAPIS, 2001).

A partir do vetor anterior, ele realiza pesquisas por novos deslocamentos ótimos, sempre conferindo o macrobloco em relação ao macrobloco imediatamente superior, ao imediatamente inferior e aos seus vizinhos laterais seguindo a direção do menor erro até atingir um dado valor limite (*threshold*). A escolha do valor de limiar, a ser adotado, pode ser fixo ou ser ajustado dinamicamente como no caso das variantes PMVFAST e EPZS (*Enhanced Predictive Zonal Search*).

Esta categoria de estimadores de busca zonal é mais eficiente que os métodos tradicionais, pois conseguem parte de um valor inicial normalmente mais próximo do resultado final (TOURAPIS et al., 2003).

5.3.2 Desenvolvimento do Módulo de Predição do Sistema

O módulo de predição entre camadas (ILP) de um codificador H.264/SVC deve se basear em informações dos vetores de camadas vizinhas a fim de buscar o maior reaproveitamento de informações e, com isso, reduzir o fluxo de dados total gerado.

Neste conceito, se ajusta muito bem o uso da estratégia de busca zonal para o módulo de predição entre camadas, uma vez que esta estratégia prevê que se utilize, conforme a necessidade, de vetores prévios, calculados por blocos vizinhos da mesma camada ou de camadas inferiores, refinando, a seguir, o resultado final, ao procurar ao redor da média dos blocos vizinhos (ponto de partida inicial) por um bloco ainda mais similar (WEI; FAN; WANG, 2009). São várias as métricas possíveis para cálculo de similaridade entre blocos, tais como erro médio quadrático (MSE – *Mean Squared Error*), PSNR, Hadamard, soma das diferenças absolutas (SAD - *Sum of Absolute Differences*), entre outras (NAM et al., 1999).

Para a presente solução foi adotada a técnica de SAD como métrica para cálculo de similaridade, por ser esta uma técnica que demanda poucos recursos computacionais (apenas subtrações, somas e operações de módulo), facilitando assim sua implementação em hardware (ZAN; AHAMAD; SWAWY, 2003). Na prática, o cálculo da métrica de SAD entre dois macroblocos é obtido pela expressão a seguir:

$$\mathrm{SAD} = \sum abs\,(PO_{ij} - PT_{ij}) \tag{24}$$

Onde:

PO_{ij} representa pixel localizado na posição (i,j) do bloco original,

PT_{ij} representa pixel localizado na posição (i,j) do bloco sendo testado.

A pesquisa por similaridade ocorre iterativamente dentro da janela de busca até se localizar o bloco mais similar. Neste caso, o bloco da imagem de referência que apresentar o menor valor de SAD (ou, em outras palavras, o menor erro residual) entre todos os blocos analisados, deverá ser selecionado como a melhor solução da predição de movimento.

Na solução desenvolvida, o módulo de predição adota um padrão de diamante pequeno para realização da busca zonal. Segundo Tourapis (2000) este tipo de topologia é vantajoso para situações de baixa movimentação ou pouca disparidade em torno do ponto de partida, convergindo de forma mais rápida e precisa para a solução do que padrões geometricamente maiores (diamante grande ou hexágono). Sendo assim, como o método de

predição proposto (busca zonal) define seu ponto de partida baseado em informações de vetores de movimento de blocos vizinhos, que normalmente mantém alta correlação entre si, a escolha deste padrão se justifica.

A Figura 97 apresenta uma ilustração do padrão de diamante pequeno citado. O circulo branco representa o ponto inicial do bloco central (figura traz exemplo de um bloco de 4x4 pixels com destaque tracejado) enquanto que os demais círculos representam blocos deslocados de uma unidade de pixel em cada direção (cima, baixo, direita e esquerda).

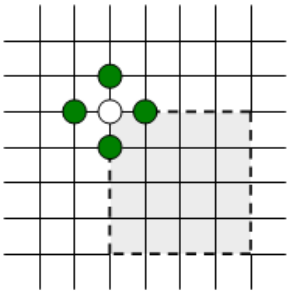


Figura 97: Padrão de diamante pequeno.

Considerando-se que o desempenho do algoritmo de predição é fundamental para a definição do desempenho global de um codificador de vídeo, ainda mais quando se trabalha com múltiplas camadas, todo ganho de velocidade possível é importante. Sendo assim, duas outras estratégias foram adotadas, como forma de reduzir ainda mais o tempo da predição: sub-amostragem e truncamento de bits menos significativos (LSB - *Least Significant Bits*).

O método de sub-amostragem procura agilizar a busca do bloco mais similar através da utilização de um número menor de amostras por bloco, ou seja, ao invés de utilizar todos os pixels do bloco de dados, se efetua uma amostragem de apenas alguns desses pixels, partindo do pressuposto que existe uma grande similaridade entre pixels próximos.

Diversos pesquisadores, tais como Liu (2000) e Porto (2008), consideraram o uso desta técnica, em seus trabalhos, avaliando experimentalmente diferentes configurações de sub-amostragem, tais como 2:1, 4:1, 8:1, entre outras. Baseado nos resultados apresentados nestes trabalhos, optou-se pela alternativa de subamostragem 4:1, onde um pixel é mantido para cada quatro amostras, isto é, um pixel a cada bloco de 2x2 pixel. Ao se adotar esta técnica, a estrutura de dados demandada para a determinação do SAD é 75% menor que a original, reduzindo o número de operações, latências de cálculo e o número de acessos à memória. Para ilustrar isto, a Figura 98 traz um exemplo de subamostragem 4:1 para um macrobloco de 16x16 pixels, em destaque, onde se observa que um pixel é extraído a cada

bloco de 2x2 pixels, gerando uma linha de pixels sub-amostrados por fileira de blocos 2x2 pixels (parte superior da figura).

Já o método de truncamento de bits procura simplificar o mecanismo de busca pela redução da resolução de cada amostra, ou seja, ao invés de utilizar todos os bits das amostras, são usados somente os bits mais significativos destas, supondo-se que a perda na qualidade por desprezar estes bits é aceitável. Porto (2008) realizou diversos experimentos práticos avaliando a perda de qualidade com diferentes configurações de truncamento de bits. Baseado nesses resultados optou-se pelo truncamento de dois bits menos significativos (descartando-se dois bits de cada amostra se obtém em uma resolução de seis bits por amostra).

Além da redução da complexidade, este mecanismo também reduz o número de acessos à memória. Na configuração adotada, até 10 amostras podem ser armazenadas em uma única posição de memória de 60 bits (cada posição contendo dez amostras de seis bits).

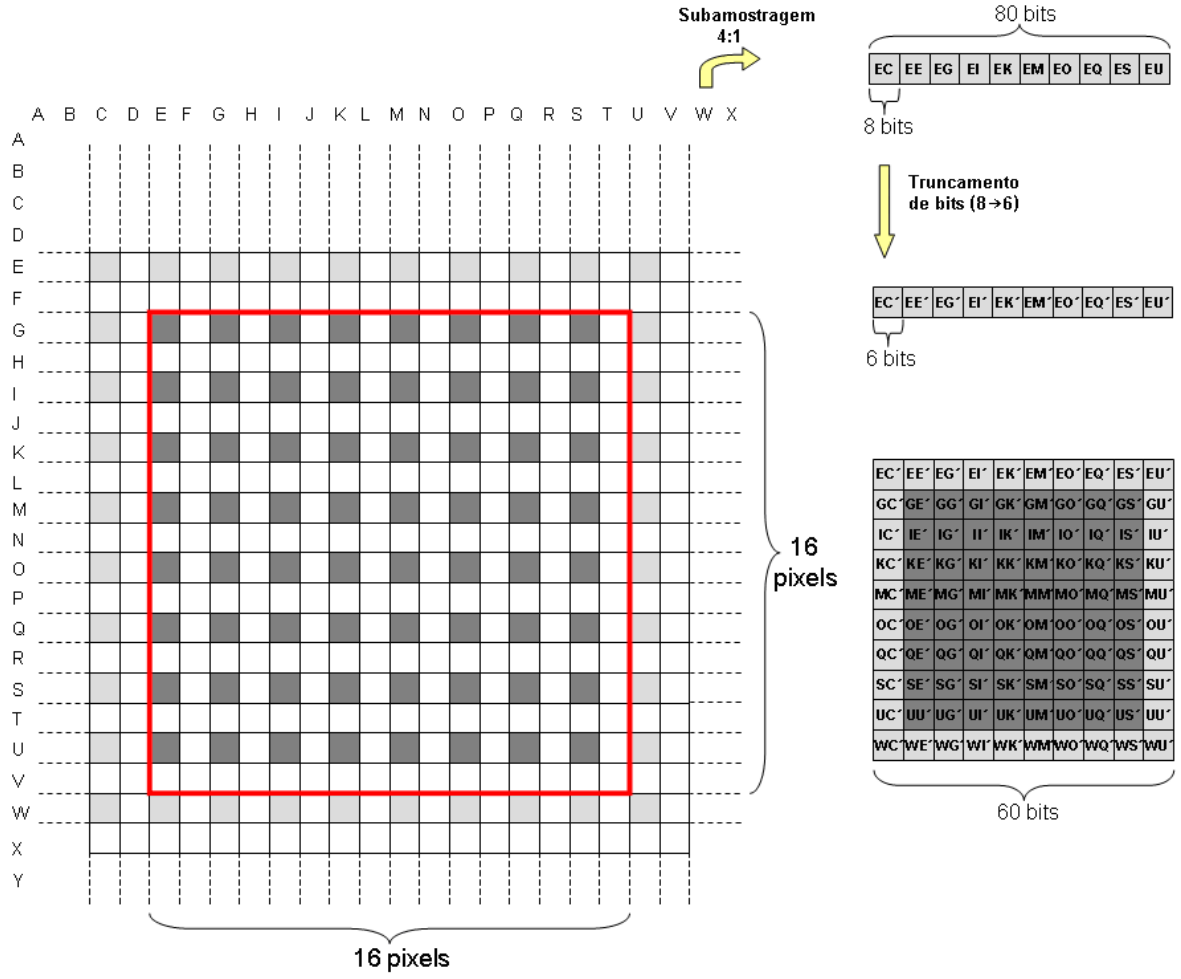


Figura 98: Esquemas adotados de subamostragem e truncamento

Estas estratégias, que em conjunto promovem o alinhamento de 10 amostras por posição de memória, além da simplificação do número de acessos à memória, contribuem diretamente para facilitar a implementação da topologia de busca em diamante pequeno, valendo-se da característica de localidade de dados. Isso se explica, pois um padrão de diamante pequeno precisa operar com o deslocamento do bloco em quatro direções possíveis (para cima, abaixo, esquerda e direita). Supondo-se, por exemplo, um macrobloco de 16x16 pixels, a amostragem de 4:1 faz com que cada linha de pixels de um macrobloco contenha oito pixels. Já para o preenchimento de um padrão de diamante pequeno completo este deve ser feito por linhas de dez pixels (oito pixels do macrobloco central mais dois adjacentes, um da direita e um da esquerda). Como cada posição de memória já armazena dez amostras de pixel esta organização é particularmente alinhada com o padrão em diamante pequeno.

A Figura 99 ilustra a relação entre um macrobloco, localizado na posição GE da memória, e os dados que acabam sendo utilizados para o cálculo de similaridade (pontos em cinza escuros correspondem ao bloco central e em cinza claros os pixels vizinhos). Cada pixel sub-amostrado na figura possui identificação alfanumérica de linha e coluna para facilitar o entendimento da relação deste com a posição original. Pode-se observar que as informações por fim consideradas pelo diamante pequeno tem dimensões compatíveis com barramentos de 64 bits, o que contribui para sua implementação prática usando memórias comerciais.

Durante o projeto, entretanto, precisou considerar uma limitação importante deste tipo de abordagem. Após a sub-amostragem 4:1, a resolução mínima para deslocamentos se torna dois pixels (resolução de blocos de 2x2 pixels) ao invés de um pixel como acontece com o mecanismo tradicional, o poderia acarretar em aumento dos erros no cálculo da predição.

A fim de evitar a incidência deste efeito, na abordagem desenvolvida, cada imagem de referência passa por processos paralelos de sub-amostragem, produzindo ao mesmo tempo distintas imagens, cada uma contendo parte dos dados originais da imagem de referência.

Mais particularmente para o caso da solução implementada (sub-amostragem 4:1) cada imagem de referência é particionada em quatro imagens sub-amostradas complementares. A idéia com esta solução é não promover o descarte de pixels, mas sim redistribuir os mesmos em distintas memórias complementares. Assim durante o procedimento de busca pelo bloco mais similar todas as imagens sub-amostradas podem ser usadas, permitindo acesso direto a qualquer posição de pixel, mesmo que, internamente, os cálculos de SAD operem com amostras sub-amostradas.

Um mecanismo de gerência próprio é responsável por selecionar e acessar a memória correta de acordo com a posição vigente da região de memória sob teste, a fim de preservar a resolução global de um pixel.

De forma geral, a arquitetura adotada para o mecanismo de predição é apresentada na Figura 99, onde se podem observar três módulos principais:

- Módulo de gerência de memórias sub-amostradas;
- Módulo de busca de bloco mais similar;
- Módulo de cálculo de SAD.

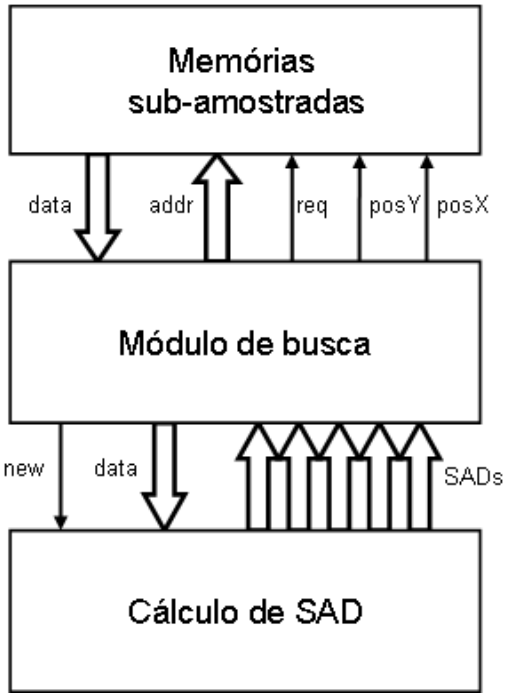


Figura 99: Visão geral da arquitetura de predição adotada

5.3.3 Módulo de Cálculo de SAD

O modulo de cálculo de SAD desenvolvido para esta solução foi especialmente projetado para suportar o cálculo do valor de similaridade de cinco posições simultâneas, seguindo a topologia de diamante pequeno.

Para tanto, esta implementação utiliza-se de uma matriz de registradores dedicada, que é preenchida a cada iteração do cálculo de SAD. A necessidade da adoção de uma estrutura baseada em uma matriz de registradores se justifica para permitir o acesso simultâneo em

posições distintas, sendo cada acesso vinculado a uma das cinco unidades internas de cálculo de SAD (memórias não foram usadas, pois só permitem o acesso de uma posição por vez).

Conforme já comentado, a técnica de sub-amostragem adotada reduz a complexidade do cálculo de SAD a um quarto do total de operações requeridas por um cálculo tradicional. O bloco de dados de referência de maior dimensão, suportado pelo padrão H.264/SVC, para o processo de predição se refere a um macrobloco de tamanho 16x16 pixels (256 amostras), o qual após a sub-amostragem 4:1 será transformado em um bloco de 64 pixels (tamanho 8x8).

Considerando-se que para o processamento paralelo do padrão de diamante pequeno completo são necessários dados adicionais de pixel nas quatro direções, chega-se assim a uma matriz de registradores com uma dimensão de 10x10 amostras.

A Figura 100 ilustra de forma genérica a relação entre imagem de referência e a matriz de registradores, quando esta é montada a partir de uma região de teste ao redor do macrobloco central de 16x16 pixels no quadro de referência. Após preencher a matriz de registradores, cinco módulos de cálculo de SAD podem ser executados em paralelo, cada um comparando o macrobloco alvo da imagem atual com um grupo distinto de pixels (ou seja, um bloco distinto de 8x8 amostras) da matriz de registradores de 10x10 amostras.

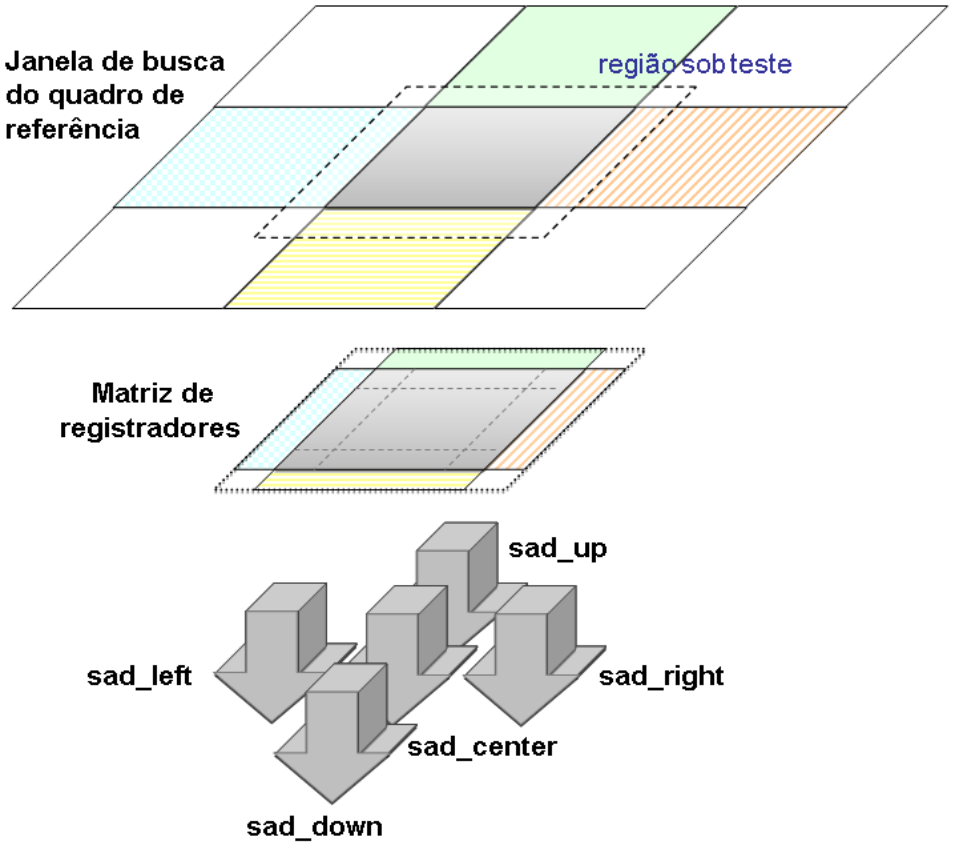


Figura 100: Mecanismo desenvolvido para predição usando diamante pequeno

Neste sentido, para a implementação desta solução foi desenvolvida uma unidade básica de cálculo de SAD, a qual foi então replicada cinco vezes para produzir os distintos valores de similaridade: *sad_center*, *sad_right*, *sad_down*, *sad_left* e *sad_up*.

A Figura 101 ilustra a arquitetura interna do módulo de cálculo de SAD, contendo cinco unidades de cálculo de SAD. A matriz de registradores é implementada por dez linhas de registradores ligadas em cascata (saída de cada registrador de linha é conectada como entrada do registrador da linha de baixo). Cada vez que um novo dado (*data_ref_in*) é fornecido ao módulo este será usado para alimentar a primeira linha da matriz.

O sinal de novo dado (*new_data_in*) comanda o processo de deslocamento de bits das diversas linhas da matriz, deslocando-se simultaneamente todas as informações da matriz uma linha para baixo. Baseado neste processo a atualização de uma matriz de registradores inteira é obtida após dez ciclos de escrita. Para cada escrita de dados a linha mais inferior da matriz é descartada. Na prática o módulo precisou implementar duas matrizes de registradores, uma para guardar a região de teste da imagem de referência e outra para armazenar o macrobloco original da imagem atual, que está sendo pesquisado. A maior diferença entre as duas matrizes de registradores é que a primeira tem tamanho de 10x10 pixels e a segunda apenas 8x8 pixels, pois precisa armazenar apenas um macrobloco subamostrado.

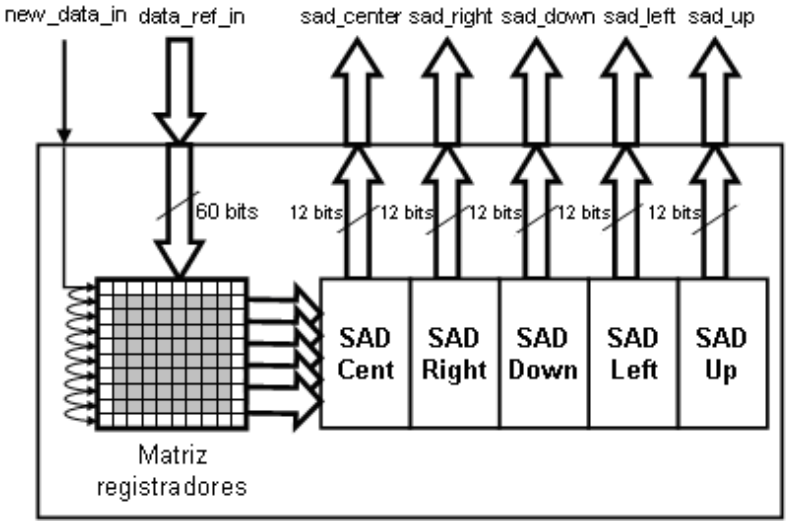


Figura 101: Estrutura interna do módulo de cálculo de SAD

As cinco unidades internas de cálculo de SAD são responsáveis por comparar, cada qual, um conjunto de 8x8 pixels do macrobloco atual com um subconjunto de 8x8 pixels da matriz de registradores de referência (cada unidade interna acessa um subconjunto distinto de forma a seguir a topologia de diamante pequeno). A implementação de cada uma dessas

unidades internas é, portanto, obtida por uma estrutura de cálculo de SAD de 8x8 amostras, que foi montada adotando-se uma estrutura de *pipeline* de apenas três estágios.

No geral cada unidade de cálculo de SAD 8x8 se baseia nos valores respectivos de quatro módulos de cálculo de SAD 4x4, que, por sua vez, é obtido a partir dos valores de quatro módulos de SAD 2x2, em uma topologia hierárquica, como mostrado na Figura 102 (alguns elementos replicados foram omitidos para simplificar a figura). Ao todo são empregadas 16 unidades de cálculo de SAD 2x2 no primeiro estágio, quatro somadores (SAD 4x4) no segundo estágio e, por fim, um somador no terceiro estágio (SAD 8x8).

A escolha desta topologia hierárquica, baseada na combinação de unidades básicas de SAD 2x2 (blocos 4x4 sub-amostrados), foi especialmente definida, de forma que a mesma solução permita a implementação do recurso de predição com particionamento de blocos, ou seja, a obtenção de vetores relacionados a blocos de tamanho variável (16x8, 8x16, 8x8, 8x4, 4x8 e 4x4), que é uma das características inovadoras da especificação H.264. Para tanto basta fazer-se o controle programável das interligações presentes do segundo estágio.

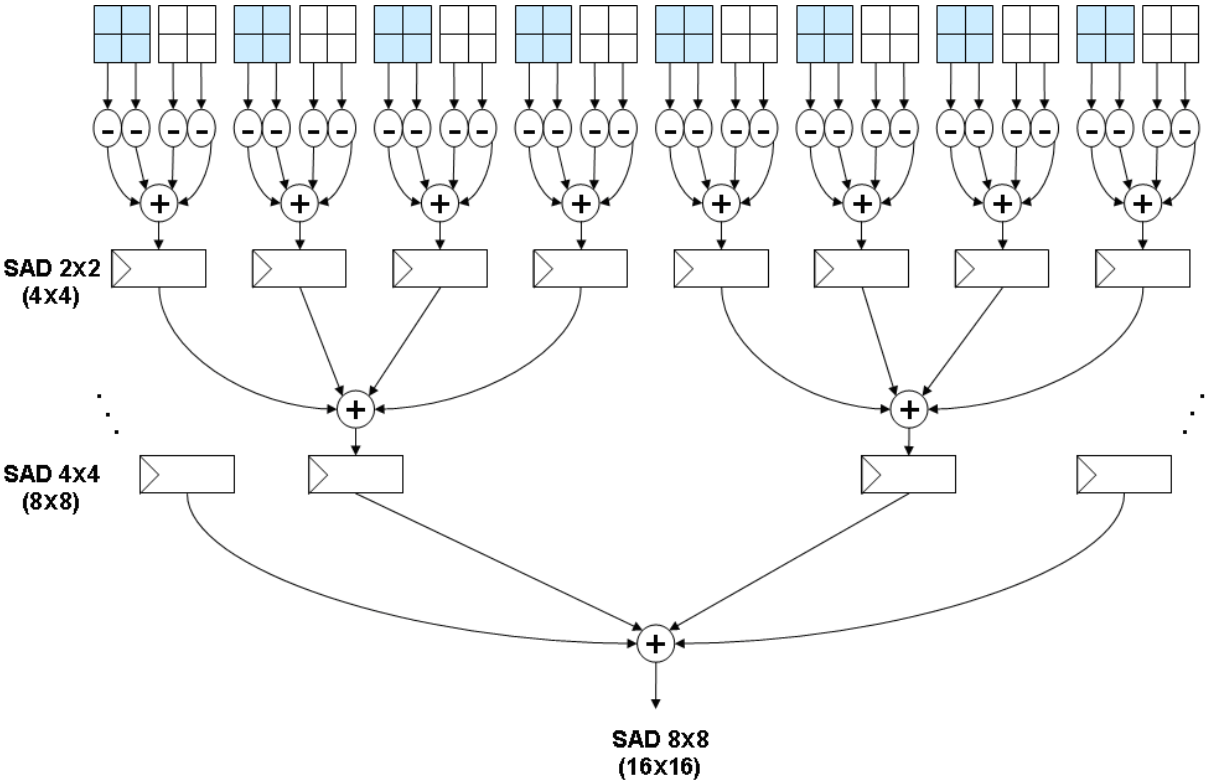


Figura 102: Implementação da unidade de cálculo de SAD 8x8

5.3.4 Módulo de Procura do Bloco Mais Similar

Este módulo é responsável por receber os cinco valores de SAD calculados no módulo anterior e, baseado nestes, decidir o próximo passo, que pode ser um novo deslocamento (para cima, abaixo, esquerda ou direita) ou simplesmente concluir o processo (quando for determinado o bloco mais similar). Antes de qualquer comparação, entretanto, o módulo precisa montar a estrutura de dados necessária para a implementação do diamante pequeno (matriz de registradores), a qual será centrada em uma posição de memória dada pelas informações obtidas da camada inferior ou mesmo de vetores obtidos por blocos vizinhos, ditos preditores.

O número de preditores adotado, bem como a localidade dos mesmos dependerá da posição do bloco atual sob teste.

Na prática, são quatro as condições possíveis do bloco no quadro de referência em relação aos seus vizinhos: (i) borda superior, quando somente o vizinho da esquerda é avaliado; (ii) borda esquerda, quando apenas o vizinho de cima é considerado; (iii) borda da direita, quando vizinhos da esquerda e superior são avaliados (média simples implementada por deslocamento de bit) e (iv) demais posições quando três preditores são considerados (média simples implementada pelo uso de tabela, para redução o tempo de cálculo).

O preditor da diagonal de cima e esquerda não é usado neste tipo de algoritmo, pois seu efeito está sendo parcialmente ponderado pelos vizinhos de cima e esquerda (TOURAPIS; AU; LIOU, 2001).

Visando facilitar o entendimento deste mecanismo a Figura 103 traz uma representação destas quatro condições citadas, onde o macrobloco atual sob teste é destacado com um bloco cinza escuro.

Um preditor adicional pode ainda ser utilizado, representando o vetor de movimento obtido na mesma posição do bloco atual, porém no quadro da camada inferior, promovendo assim o real conceito predição entre camadas. Entretanto a utilização ou não deste preditor em especial dependerá de configuração do mecanismo de escalabilidade a ser adotado (uma vez que está é uma das diferenças das estratégias CGS e MGS).

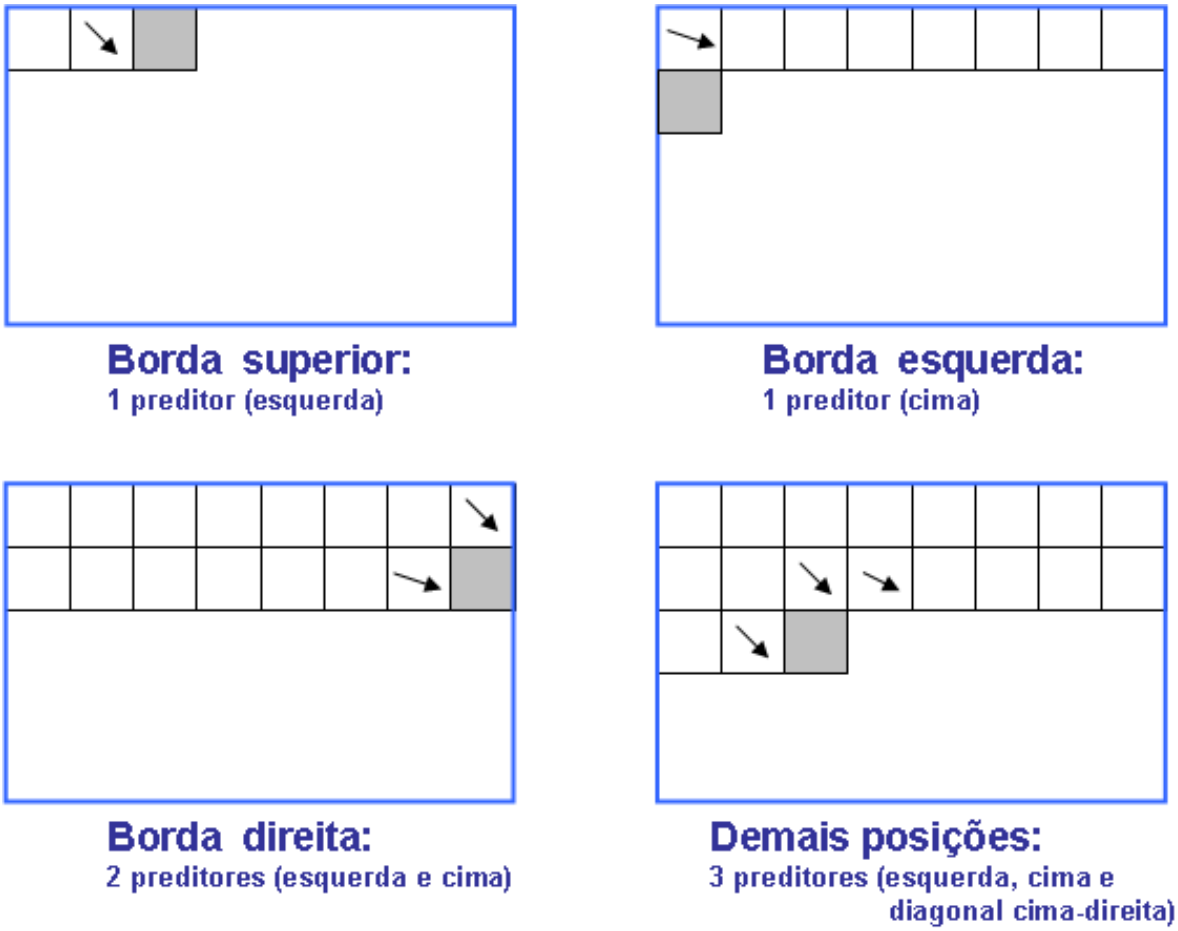


Figura 103: Preditores vizinhos considerados conforme posição do bloco

O resultado da média dos preditores é uma informação de deslocamento relativo, que deve então ser somada à posição do bloco atual sob teste para se determinar a posição inicial da imagem de referência, que será utilizada para montar a matriz de registradores interna do módulo de cálculo de SAD (diamante pequeno).

O alinhamento particular de dados adotado, após sub-amostragem e truncamento de bits, reduz consideravelmente o tempo gasto com acessos à memória pelo módulo de predição.

Na condição ideal (bloco a ser buscado estando corretamente alinhado com a organização da memória de referência), a leitura de uma linha inteira de dez pixels ocorre com somente um acesso à memória.

Neste caso, o preenchimento completo da matriz de registradores do módulo de cálculo de SAD é obtido após dez acessos em uma memória de 64 bits, o que representaria uma latência de apenas dez ciclos de relógio para a conclusão desta operação (evidentemente considerando-se o uso de memórias síncronas).

No pior caso, quando o endereçamento de memória estiver desalinhado com a posição de bloco inicial, dois acessos à memória são necessários para a determinação de cada linha da matriz (uma vez que parte dos dados está em uma posição de memória e o restante na posição consecutiva), gerando assim um total de vinte ciclos para o preenchimento completo da matriz.

A Figura 104 ilustra um exemplo deste procedimento, quando a posição absoluta calculada com a ajuda dos preditores foi (3, 6).

Na parte superior da figura se observa, em destaque, a identificação do ponto da memória de referência onde se deve começar a ler os dados para atualizar a matriz de registradores.

A variável N representa o número de posições ocupadas para armazenar uma linha de pixels da imagem na memória de referência. Assim sendo, cada vez que se desejar descer uma posição (um pixel) da imagem deve-se incrementar o endereço da memória de referência de N posições.

A posição calculada no exemplo, posição (3, 6), está desalinhada com a memória. Por isso são necessários dois acessos para cada linha a ser montada. No centro da figura o mecanismo adotado para esta montagem é apresentado. O primeiro acesso ocorre na posição 3xN (quarta linha de pixels da imagem).

Apenas os quatro pixels inferiores da informação obtida com esta leitura serão usados para a montagem do vetor de pixels desejado. Assim sendo um novo acesso deve ser feito, a seguir, na posição consecutiva da memória (endereço 3xN+1), quando são obtidos os seis pixels restantes para compor o vetor de 10 pixels desejado.

O processo deve se repetir por dez vezes, a cada iteração lendo-se dois endereços consecutivos e montando-se cada vetor desejado, que é então enviado ao módulo de cálculo de SAD. Ao término deste procedimento (20 ciclos de relógio) a matriz de registradores estará, por fim, preenchida com todos os valores desejados.

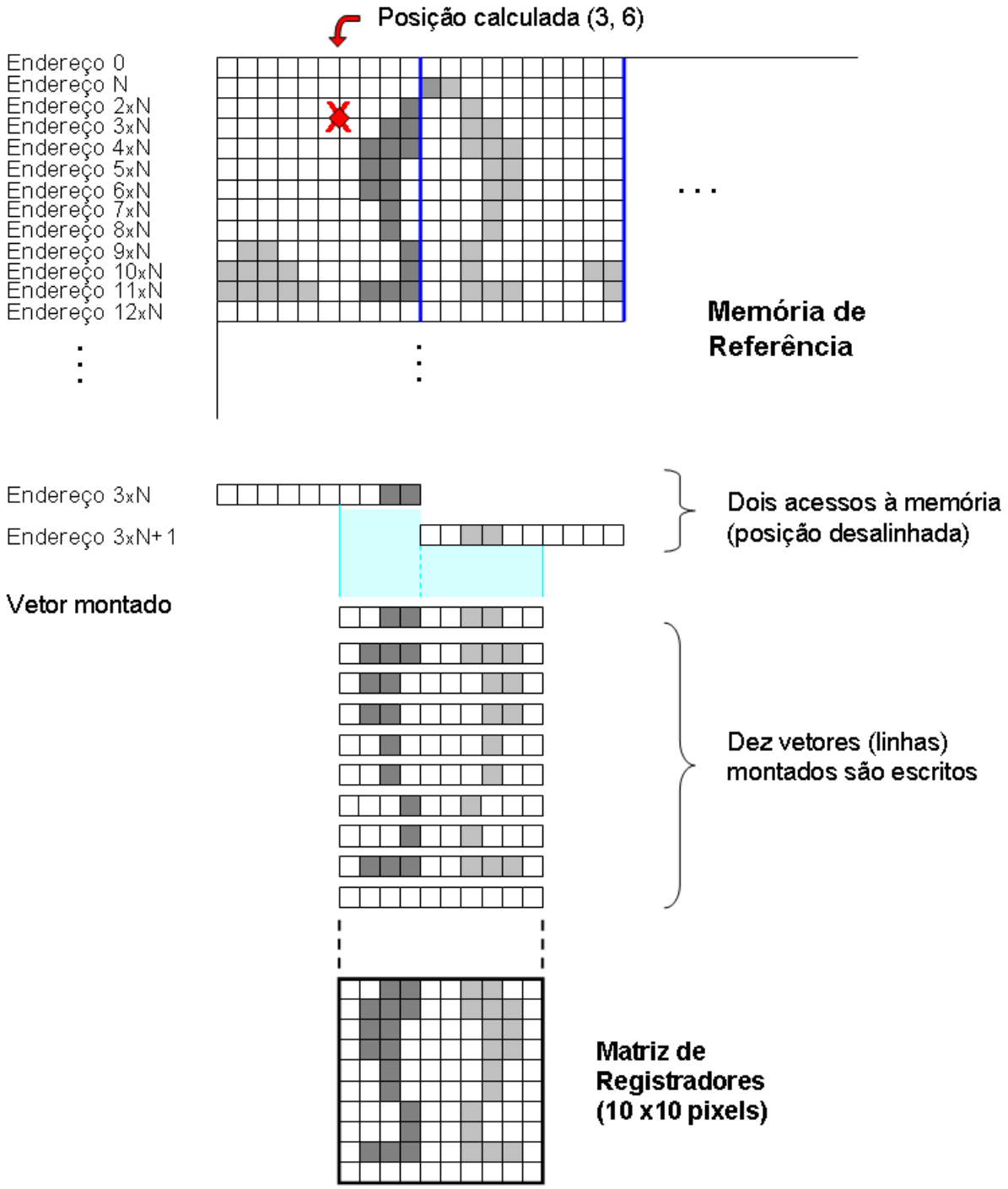


Figura 104: Utilização de memórias alinhadas horizontalmente

Após o correto preenchimento da matriz de registradores, o módulo de busca deve aguardar mais três ciclos de relógio (latência do módulo de SAD) e, a seguir, avaliar os cinco valores de SAD obtidos.

A tomada de decisão considera a topologia de padrão de diamante pequeno e pode iteragir por várias etapas, dependendo da distância do bloco mais similar em relação ao ponto de partida.

Na primeira etapa da busca, a similaridade do bloco central é comparada com os demais quatro valores ao redor deste. O valor de SAD que obtiver o menor resultado de erro absoluto indica o caminho para o próximo estágio. Pode-se, desta forma, obter quatro opções de continuidade para a segunda etapa de comparação (Figura 105). Se o bloco superior for o que possui menor SAD, deve passar para uma busca em padrão de diamante pequeno centrado neste bloco. O mesmo vale para os demais blocos (direita, esquerda e abaixo). Na parte superior da Figura 105, aparece a etapa anterior de comparação e abaixo as quatro direções que podem ser tomadas na próxima etapa (abaixo, direita, acima e esquerda respectivamente). Os quadrados representam as possíveis novas topologias de diamante pequeno montada.

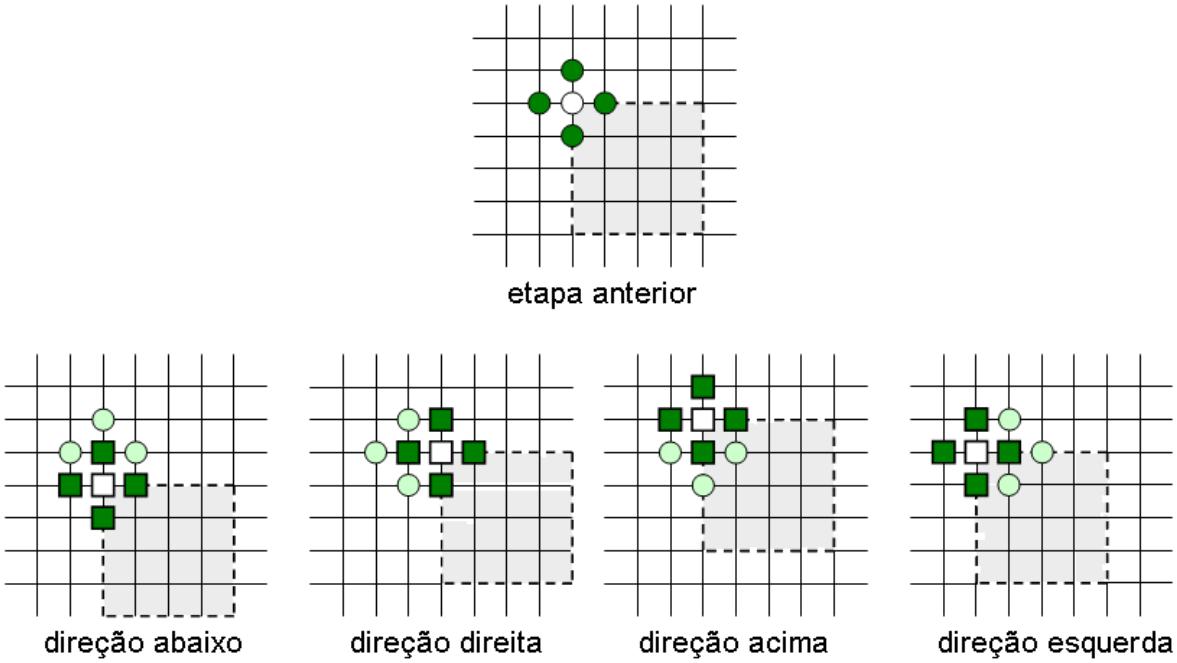


Figura 105: Direções de movimentação sobre o padrão de diamante pequeno.

A cada novo passo, a matriz de cálculo de SAD deve ser atualizada e novo procedimento de comparação deve ser feito, sempre deslocando a busca na resolução de um pixel. Na prática isto é feito simplesmente incrementando-se os índices de posição vertical ou horizontal da imagem.

Se o menor SAD determinado em qualquer estágio estiver localizado no bloco central do diamante pequeno, a busca se encerra nesta etapa e o resultado obtido é a diferença da posição deste último bloco central e a posição real do bloco sob teste.

Para o caso de dois ou mais valores de SAD extremos apresentarem valores inferiores ao central, porém iguais se optará pelo bloco que estiver mais alinhado com a direção média dos preditores (obtida no início do procedimento).

Já para o caso particular de o valor erro absoluto em qualquer direção, não for menor, porém igual ao do bloco central, o algoritmo tentará seguir na direção desse bloco de extremidade ao invés de parar.

Esta decisão é tomada a fim de se oportunizar a procura de um novo vetor, possivelmente melhor. Mas isso acontecerá apenas se a última opção de deslocamento adotada não for oposta a esta nova descoberta. Ou seja, se for detectado que o SAD do bloco da direita é igual ao central, o próximo passo seguirá na direção da direita. Assim na próxima etapa (após se deslocar para a direita), os blocos da esquerda e central deverão ser iguais, uma vez que, nesta nova etapa, o bloco da esquerda representa o antigo bloco central e o novo bloco central representa o antigo bloco da direita. Nesta condição, o algoritmo não tentará retornar a procura pela esquerda, mesmo encontrando estes dois valores iguais, pois já estava vindo de lá (último deslocamento foi da esquerda para a direita).

Algoritmos como este, que realizam deslocamentos na tela em topologias geométricas, dependendo dos valores gravados na memória de referência, estão sujeitos a ficarem presos em laços de repetição involuntária, caso em que não ocorre convergência para um bloco. Uma forma prática de proteção a este problema consiste em limitar o número máximo de iterações por busca.

5.3.5 Módulo de Memórias Sub-amostradas

O módulo de memórias sub-amostradas é responsável pela manutenção das áreas de janela de busca necessárias para realização do procedimento de predição, que se distribuem internamente em quatro memórias sub-amostradas, de forma complementar.

Para seu preenchimento o módulo busca dados de uma memória externa informações do quadro de referência (imagem recuperada), a fim de compor as quatro memórias subamostradas, internamente referenciadas como *Ref_PP* (contendo pixels de linhas e colunas

pares), *Ref_PI* (linhas pares e colunas ímpares), *Ref_IP* (linhas ímpares e colunas pares) e *Ref_II* (linhas e colunas ímpares). Para ilustrar isto, a Figura 106 descreve a estrutura interna do módulo desenvolvido de gerência de memórias sub-amostradas.

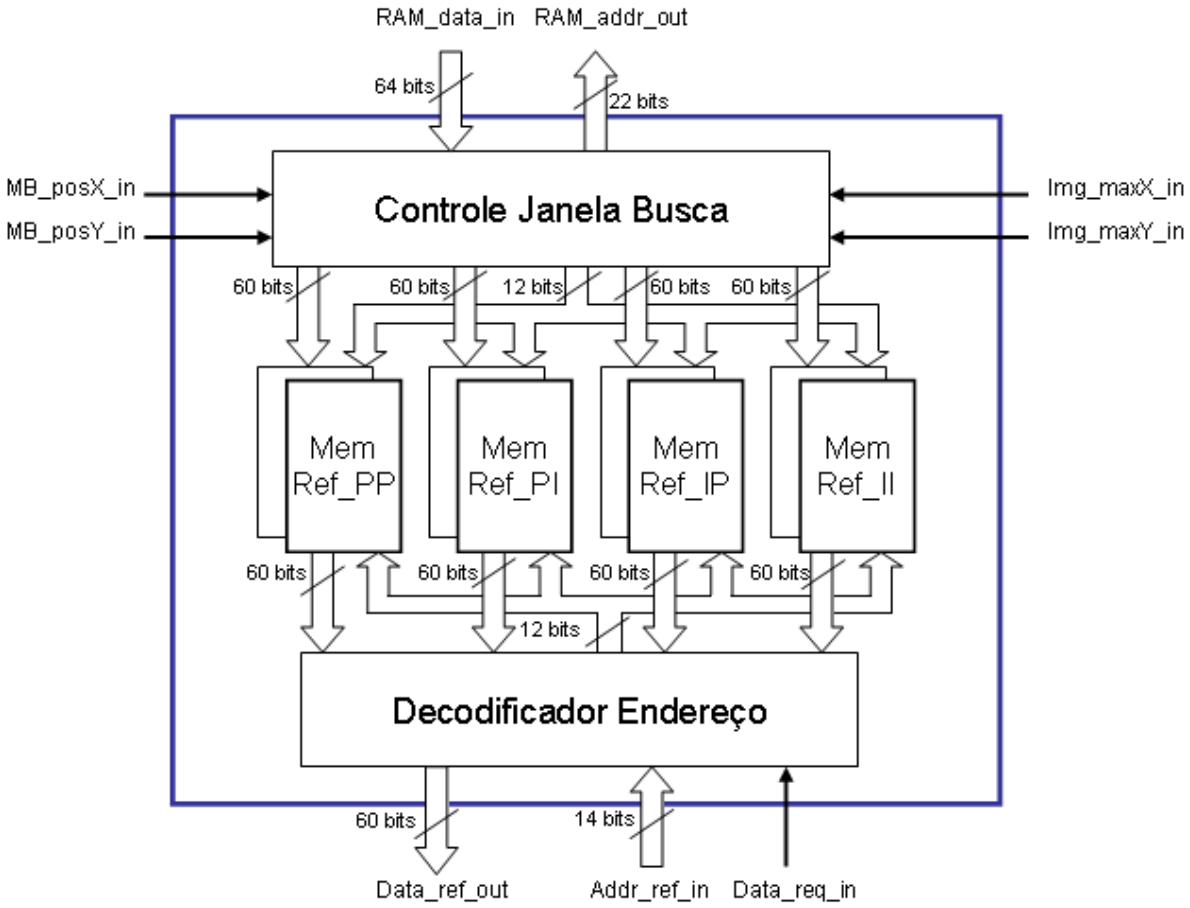


Figura 106: Visão geral do módulo gerente de memórias sub-amostradas.

A parte superior da figura descreve a entidade responsável pela interface com a memória externa e atualização das quatro memórias complementares (“Controle Janela Busca”).

Pode-se observar que para controlar o correto preenchimento da janela de busca esta entidade precisa se valer de algumas informações externas: *MB_posX_in* e *MB_posY_in*, que definem a posição absoluta do macrobloco atual sob teste (referências horizontal e vertical respectivamente), e *Img_maxX_in* e *Img_maxY_in*, que definem as dimensões físicas do quadro de referência (também nas direções horizontal e vertical respectivamente).

Visando aumento de desempenho do módulo, pelo menos dois bancos de memórias sub-amostradas devem ser utilizados para a montagem desta solução. Assim, enquanto a

região gravada em um dos bancos estiver sendo utilizada pelo módulo de busca (módulo "Decodificador Endereço" na parte inferior da figura), outro banco estará, em paralelo, sendo preenchido com dados de outra região ("Controle Janela Busca" na parte superior).

Quando a busca for completada em um dado banco, o módulo pode simplesmente ser chaveado para o outro banco já preenchido, liberando o banco antigo para nova atualização. O chaveamento entre bancos acontece, de forma automática, apenas avaliando-se os bits da posição acessada.

A idéia presente no projeto deste módulo foi que seu funcionamento interno deveria ser transparente para módulos externos. Ou seja, os módulos externos não necessitam conhecer detalhes da construção e distribuição das memórias internas.

Do ponto de vista funcional, o módulo externo só precisa atualizar o respectivo endereço de memória desejado (*Addr_ref_in*), que o módulo de memórias sub-amostradas automaticamente irá apresentar o dado referente a esta posição em *Data_ref_out* (com largura de 60 bits).

Internamente, porém, este módulo de gerência de memória precisa verificar, a partir da posição requisitada, qual endereço e de qual memória deverá ser acessada para atender ao pedido externo.

Na prática, a operação ocorre pela entidade "Decodificador Endereço". Sempre que o acesso for a uma coluna par e uma linha também par, o pedido será encaminhado à memória *Ref_PP*. O mesmo princípio vale para as demais memórias (*Ref_PI*, *Ref_IP* e *Ref_II*).

Como forma de reduzir a lógica de acesso às memórias internas, foi implementada com uma estratégia de alinhamento bidimensional, o que quer dizer que o endereço é constituído pela concatenação de duas informações distintas de posição (linha e coluna do dado desejado).

A redução de lógica acontece pela associação direta do endereço com informações internas do módulo de busca de bloco mais similar, que opera atualizando, a cada etapa, as informações de linha e coluna da posição de memória a ser acessada.

Um melhor entendimento desta solução pode ser obtido observando-se a Figura 107, que demonstra o procedimento adotado para leitura das quatro memórias sub-amostradas internas.

Na solução adotada todas as memórias são acessadas simultaneamente, sendo um multiplexador de 4x1 usado para selecionar na saída qual delas deve ser de fato considerada.

Como comentado antes, o endereço de memória desejado é obtido pela concatenação das informações respectivas de coluna e linha (na figura, identificadas pelos nomes de "*Endereço Lin*" e "*Endereço Col*"). Neste contexto o bit menos significativo de cada uma destas informações pode ser usado, como forma prática, para identificar se o acesso a ser feito ocorre em uma posição par ou ímpar. Assim, sendo a seleção pela memória correta, é realizado, de forma simples, utilizando-se para controle deste multiplexador o bit menos significativo das posições de coluna e linha (dois bits de controle para um MUX 4x1).

Utilizando esta estratégia, quando um deslocamento para a direita for requerido pela estrutura de diamante pequeno o módulo de busca de bloco mais similar somente precisa incrementar a posição de coluna a ser acessada. Isso faz com que o bit menos significativo inverta. Assim se a última memória selecionada foi a *Ref_PP*, a partir do incremento, o controle do multiplexador 4x1 faz então que seja acessada a memória *Ref_PI*.

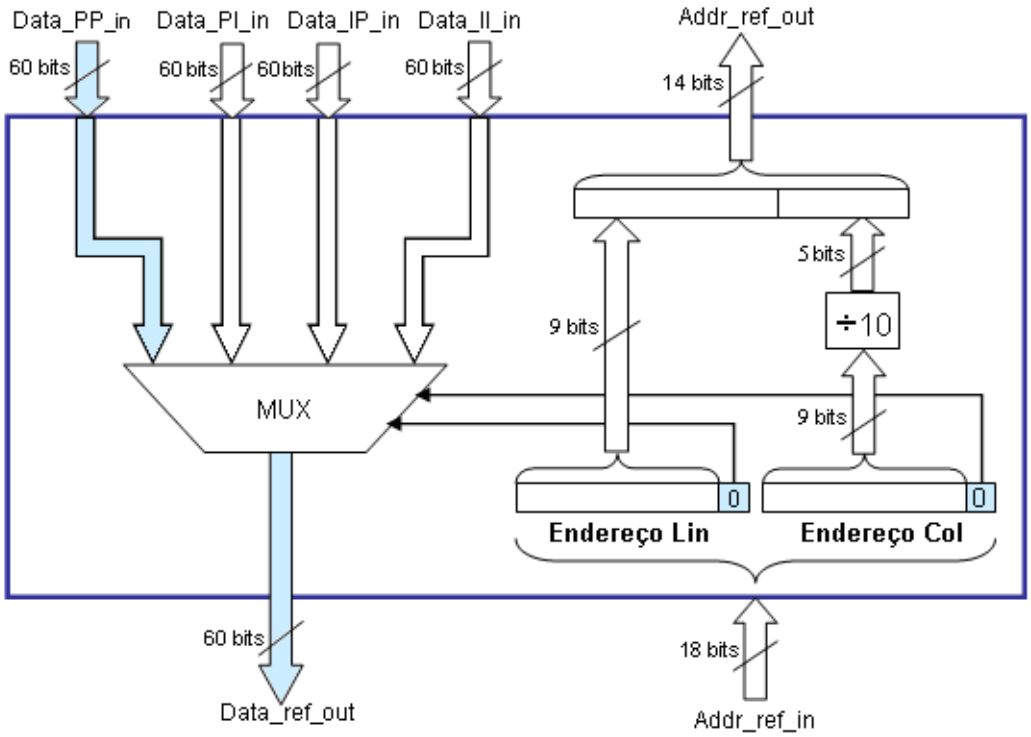


Figura 107: Procedimento de leitura das memórias sub-amostradas.

Observando-se a Figura 107 pode-se perceber adicionalmente a presença de um módulo de divisão por dez que atua sobre a posição horizontal ("*Endereço Col*"). Este módulo é necessário, pois as memórias empregadas na solução, com largura física de 64 bits, contém, em cada uma de suas posições, a informação de dez pixels consecutivos orientados

em linha. Assim, muitas vezes, apesar de se realizar deslocamentos horizontais no módulo de busca as posições a serem acessadas na memória se manterão as mesmas. De fato a variação de endereçamento em memória se processará somente variando, em módulo de 20 incrementos de pixels na posição horizontal (correspondentes a 10 pixels de memória par ou ímpar).

A implementação deste módulo de divisão em hardware foi feita através de uma tabela em ROM para permitir seu cálculo em um único ciclo de relógio.

6 RESULTADOS EXPERIMENTAIS

6.1 CARACTERÍSTICAS DO AMBIENTE DE TRABALHO

Para fins de validação, a presente arquitetura foi implementada em uma aplicação real de codificação de vídeo H.264/SVC. Mais particularmente foi montada uma solução colaborativa onde um computador pessoal executa uma versão adaptada de codificador em software, sendo apoiado em tempo de execução por módulos de hardware, sintetizados sobre uma placa de desenvolvimento que utiliza a tecnologia de FPGA.

O computador escolhido foi um Pentium i7-860, com 4 núcleos de processamento, operando a uma freqüência de 2,8GHz, com memórias cache L1 (32Kbytes +32Kbytes), L2 (256Kbytes +256Kbytes) e L3 de 8Mbytes, 3Gbytes de memória DDR3 1333MHz (duplo canal) e 500GB de disco rígido 7200rpm. Sobre este computador foi instalada uma versão refinada do software de referência JSVM versão 9.19 (JOINT SCALABLE TEAM MODEL, 2010), do grupo JVT (detalhes desta versão foram apresentados na seção 4.2.1.7).

A plataforma utilizada para a implementação da parte de hardware foi a placa de desenvolvimento XUPV5-LX110T. Entre seus recursos principais, esta placa contém um dispositivo Xilinx Virtex-5 XC5VLX110T com capacidade de 110.000 células lógicas, interface para cartões Compact Flash, 256 Mbytes de memória DDR2 de 64 bits com taxa de dados de 400MHz, além de 9 Mbits de memória síncrona ZBT SRAM de 32 bits com tempo de acesso de 5ns (200MHz), interface PCI Express 1x em padrão de slot para computador, uma interface Ethernet 10/100/1000 e controlador USB v2.0 (XILINX, 2010).

Confrontando as características desta placa com os requisitos de memória e interface avaliados na Seção 4.2.4 pode-se, a princípio, afirmar que, teoricamente, esta placa atende as demandas mínimas exigidas para uso nesta solução, quanto às capacidades físicas de memória e interface de comunicação. Para ilustrar esta placa, a Figura 108 traz uma fotografia da mesma.

Pode-se observar, no centro da placa, a presença do dispositivo de lógica programável principal (Virtex 5 XC5VLX110T) e, logo abaixo, a interface para barramento PCI Express.

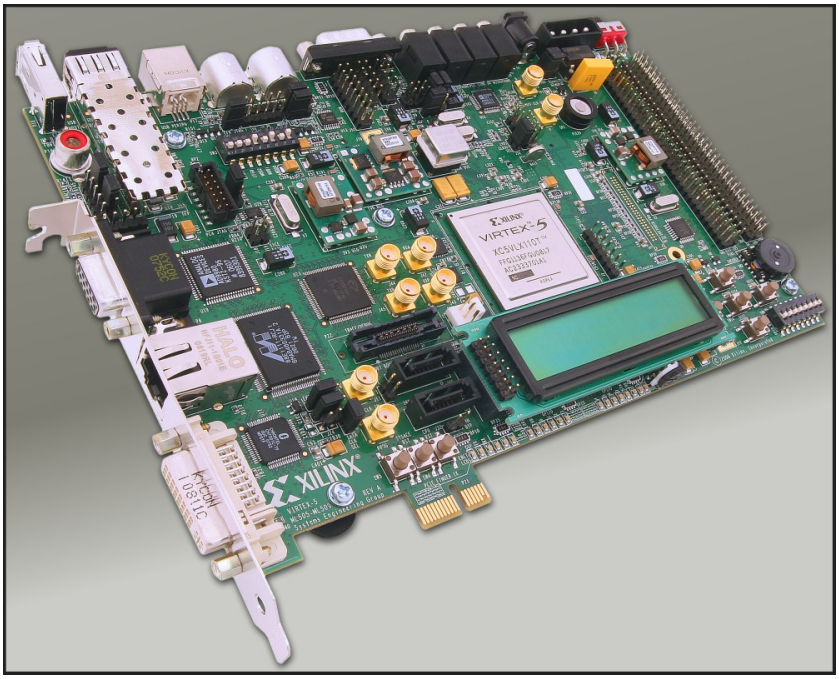


Figura 108: Fotografia da placa utilizada para experimentos práticos

A entidade de software foi implementada tomando-se como base o código de referência H.264/SVC oficial, JSVM versão 9.19.4, que é mantido e disponibilizado pelo grupo JVT, entidade responsável pela especificação e divulgação do padrão. Para ser utilizado neste projeto, o software de referência JSVM passou por uma intensiva avaliação visando identificar as estruturas de dados e interfaces necessárias para integração dos diversos módulos envolvidos. De forma simplificada, o funcionamento do software JSVM está baseado em quatro bibliotecas (JOINT SCALABLE VIDEO MODEL, 2010):

a) *H264AVCEncoderLibStatic*: conjunto de classes com todas as funções referentes à codificação de vídeo propriamente dita, tais como módulo de predição entre camadas, algoritmo de controle de taxa, entre outras;
b) *H264AVCDecoderLibStatic*: conjunto de classes com funções relacionadas à decodificação de vídeo, tais como decodificador de entropia e de cabeçalho;
c) *H264AVCCommonLibStatic*: conjunto de classes comuns para diferentes aplicações, transformadas, quantizações e filtragem;
d) *AvcRewriterLibStatic:* conjunto de classes usadas para a manipulação de fluxos escaláveis, visando configuração de diferentes parâmetros, tais número de camadas, tipos de escalabilidade, relações de proporção, entre outras.

A utilização destas bibliotecas distintas visa aumentar a modularidade do software, agrupando funções próprias em pacotes de software fechados que possam ser incorporados a outros projetos. Assim a construção de um decodificador escalável precisa apenas incluir as bibliotecas *H264AVCDecoderLibStatic* e *H264AVCCommonLibStatic*. Detalhes de implementação internos do código podem ser abstraídos pelo usuário, que precisa apenas fazer chamadas a funções definidas como interface da biblioteca.

Após a análise deste software, foram identificados os arquivos e métodos específicos a serem alteradas dentro das bibliotecas diretamente envolvidas: *H264AVCCommonLibStatic*, onde se encontram os módulos de transformadas, quantização e filtragem, e *H264AVCEncoderLibStatic*, onde fica localizado o módulo de predição entre camadas.

O software JSVM utiliza a linguagem de programação C++, explorando o conceito de orientação a objetos. As estruturas de dados utilizadas para operação passam por classes especialmente desenvolvidas, que mantém diversas informações importantes, tais como posição de macrobloco, tipo de componente (luminância ou crominância), tipo de codificação (intra ou inter), ponteiro para área de dados original, ponteiro para área de dados recuperados, entre outras.

A solução adotada manteve a mesma organização, apenas acessando estas classes para buscar os dados a serem enviados para a placa, bem como atualizando os mesmos após os mesmos terem sido processados e retornados.

6.2 DESCRIÇÃO DOS EXPERIMENTOS DESENVOLVIDOS

A estratégia de validação experimental do projeto passou por três etapas distintas:

a) Avaliação prática da plataforma de trabalho;
b) Validação das funcionalidades individuais de cada módulo novo;
c) Integração final dos módulos operacionais.

Na primeira etapa, a plataforma de trabalho foi testada de forma genérica avaliando-se principalmente os tempos envolvidos com os processos de comunicação entre as entidades de hardware e software. As informações obtidas nesta etapa serviram para refinar o modelo de integração.

Em um segundo momento, os módulos em hardware foram implementados um por vez, de forma a garantir a validação individual de seu funcionamento. Nesta etapa, os módulos de softwares foram alterados gradualmente, conforme os módulos de hardware eram concluídos, sem a preocupação imediata com o desempenho obtido.

Como estratégia de validação, antes de executar cada algoritmo em software os dados originais eram enviados para a placa via chamada de funções ao barramento PCI-Express (*driver* de comunicação). Na placa os dados enviados eram processados pelos módulos de hardware sob teste e a seguir devolvidos para o software. Os dados eram, a seguir, processados em software e comparados com os dados produzidos pelo hardware para validação de consistência. Somente após a validação individual de cada módulo, estes eram integrados para a geração de protótipos funcionais.

Somente em um terceiro momento foram implementadas as integrações de fato, onde módulos em software foram substituídos por módulos em hardware. Dois protótipos foram gerados: um de codificação intra, responsável pela geração de quadros I (sem algoritmos de predição) e um de codificação inter, quando foi introduzida a funcionalidade de predição entre camadas (quadros P e B).

O software no computador ficou responsável pelas tarefas de interface com o usuário, leitura do vídeo original, entropia, controle de taxa e geração do fluxo de saída final.

Na seção a seguir se apresentam os resultados obtidos com os diferentes ensaios realizados. Inicialmente são avaliadas as características próprias da plataforma de trabalho e latências envolvidas com o processo de comunicação entre hardware e software. A seguir são apresentados os resultados obtidos com as integrações, seja avaliando os módulos individualmente, seja em conjunto.

6.3 INTEGRAÇÃO COM O SOFTWARE JSVM

A integração do software de referência JSVM com os módulos em hardware aconteceu a partir de um *driver* PCI Express, que é disponibilizado pela própria empresa Xilinx. Este software é feito para operar em conjunto com solução Xilinx LogiCore PCIExpress, permitindo a conexão de dispositivos da Família Virtex 5 com esta interface de comunicação (XILINX, 2005).

A implementação de interfaces PCI Express em dispositivos de FPGA foi particularmente facilitada pelo lançamento das recentes plataformas da Xilinx (famílias de

FPGA Virtex 5 e Virtex 6) que fornecem os blocos internos dedicados com esta finalidade, capazes de executar todo o tratamento de baixo nível da interface PCIe (sincronização de relógio e serialização dos dados). Cada bloco de hardware destes pode ser usado para executar um canal independente, denominado de *Endpoint* PCI. A estrutura deste módulo mantém os registradores de configuração necessários para possibilitar o recurso de Plug&Play da interface PCIe, fazendo comunicação com os demais módulos internos através de memórias do tipo FIFO de 64 bits. Uma representação deste módulo é apresentada na Figura 109. Por se tratar de um módulo proprietário da empresa Xilinx seu princípio de funcionamento será desconsiderado para este projeto. Maiores detalhes podem, adicionalmente, serem obtidos em (XILINX, 2010).

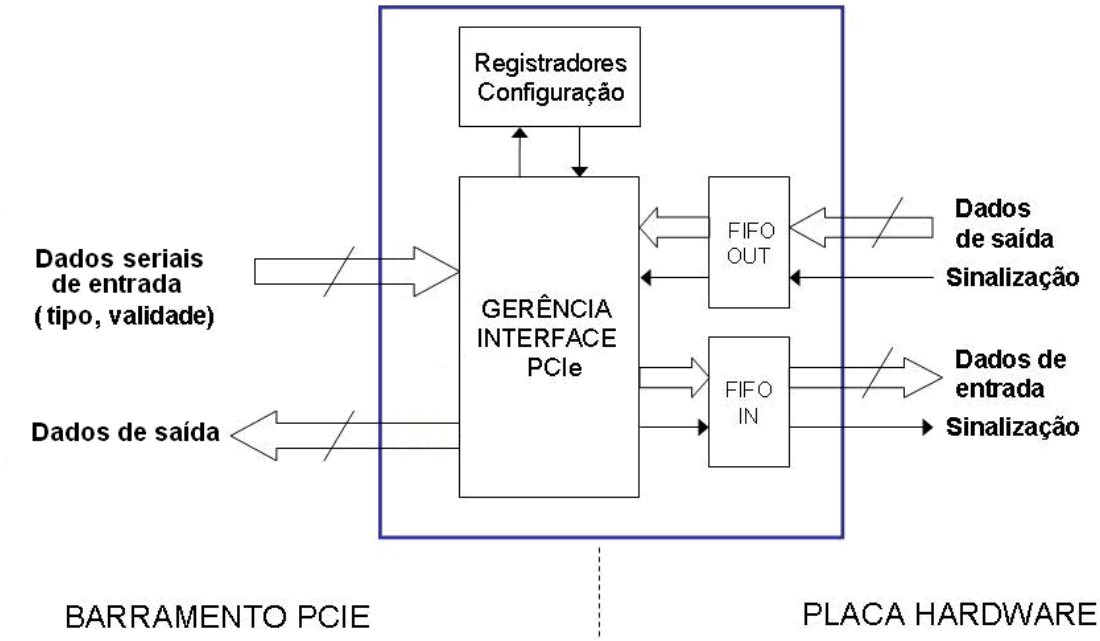


Figura 109: Módulo de interface PCIe Xilinx LogiCore

As transmissões de dados por um canal PCIe são divididos em pacotes de transação predefinidos pelo protocolo, que são chamados de TLPs (*Transaction Layer Packets*). Toda e qualquer mensagem PCIe precisa ser definida em unidades de TLP, sendo que a quantidade de dados presentes em uma TLP pode ser configurada.

A fim de definir o melhor valor para este parâmetro, foram avaliados os resultados de taxa efetiva de dados obtidos transmitindo-se dados para a placa. Para este experimento, foi instalada na placa uma aplicação de replicação de pacotes (todos os dados recebidos eram enviados de volta ao computador sem processamento algum).

A Figura 110 apresenta os resultados obtidos com este ensaio, para diferentes tamanhos de TLP, onde se percebe que o melhor resultado para TLPs com tamanho de 32 palavras duplas (32x32 = 1024 ou 128 bytes por TLP).

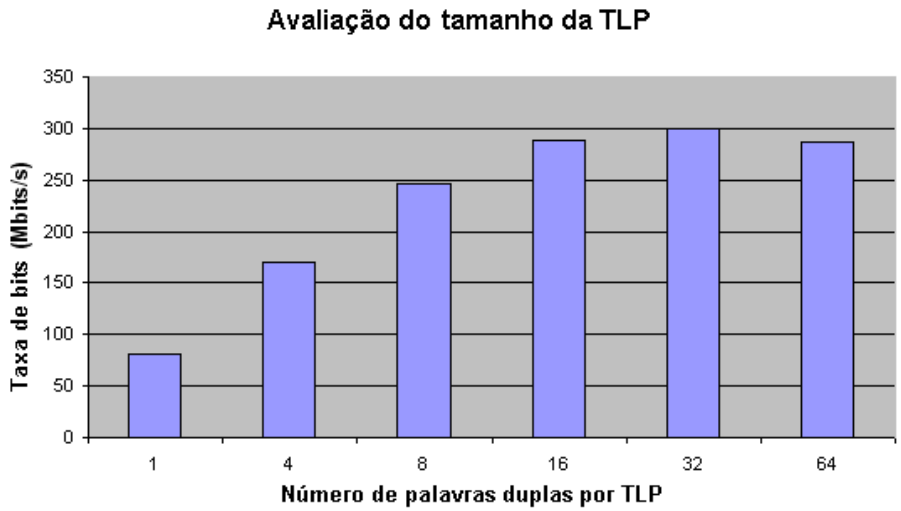


Figura 110: Resultados da análise do tamanho da TLP x taxa efetiva de dados.

Os resultados se explicam pelo fato de que os pacotes de dados enviados e recebidos são internamente divididos em pacotes de dados e configuração, ou seja um pacote de dados, quando muito extenso, é dividido internamente pelo sistema de comunicação em diversos pacotes menores, o que acaba reduzindo a flexibilidade da interface a partir de certo ponto (TLP = 32 palavras duplas). Este efeito pode ser observado na Figura 112.

Transferências PCIe utilizam nativamente o recurso de DMA (*Direct Memory Access*), que procura otimizar a taxa de dados ao permitir a transferência direta de dados entre *chipset* de comunicação e a memória da placa. Mesmo assim a taxa efetiva na placa não é contínua, por limitações da interface (PCIe 1x) e pela necessidade de se quebrar periodicamente as mensagens em pacotes menores (TLPs).

Esta característica foi observada na prática durante os primeiros ensaios de integração. Essa verificação foi possível pelo uso da ferramenta de monitoração Chipscope Pro Analyzer tool da empresa Xilinx (AKPAN, 2009), que permite a captura em tempo de execução de sinais internos da implementação.

Pode-se ver claramente na Figura 111 como, durante um processo de transferência PCI capturado, existem momentos em que o barramento de dados é, de fato, ocupado, enquanto que, em outros, o mesmo fica ocioso.

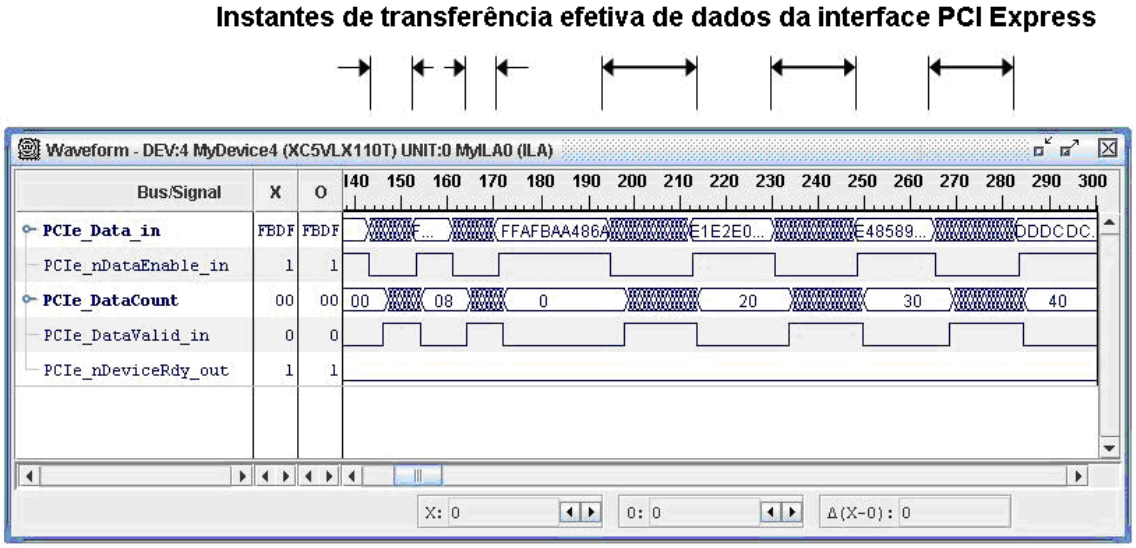


Figura 111: Sinais do barramento capturados durante uma transferência PCI.

De forma geral foi medido, neste experimento, que uma transferência de 2048 bytes se processa em um intervalo de 25µs, o que corresponde a uma taxa efetiva máxima de 135,9Mbytes/s ou 1087 Mbit/s. Este dado indica que a interface da placa, de fato, restringe a velocidade de acesso para um valor bem abaixo da máxima taxa teórica de 2000bit/s, o que impõe a esta plataforma restrições para atender aos requisitos teóricos definidos na Seção 4.2.4.

A explicação para esta perda de desempenho se explica por dois motivos: (i) a quebra da mensagem PCI-Express em diferentes TLPs, que levam a tempos perdidos entre pacotes (o que pode ser observado na Figura 112) e (ii) as latências do barramento de comunicação, que acontecem deste o pedido de uma escrita na placa até a escrita efetiva e também na volta, desde o preenchimento dos dados processados e o retorno em si.

Este tempo de latência depende de inúmeros fatores, tais como: escalonador de tarefas do sistema operacional, forma de implementação do *driver* de comunicação, freqüência de trabalho do computador, entre outros.

Visando-se avaliar praticamente esta latência, foi desenvolvido um módulo simples de verificação na placa em hardware. A função deste módulo é a de iniciar um contador quando este perceber o primeiro dado na interface de entrada. A cada ciclo de relógio, a partir daí, o contador é incrementado. Enquanto isso, os dados então eram lidos e copiados para a interface de saída (retorno para o aplicativo no computador). Ao final do último dado escrito o valor do contador é escrito e anexado ao pacote. Assim pode-se determinar quanto tempo do atraso

total da comunicação se passou dentro da placa. A diferença destes valores representa o tempo de latência da comunicação.

Os resultados obtidos são apresentados na Tabela 11.

Tabela 11 – Medidas de latência de comunicação

Experimento	Tempo total de comunicação	Tempo de comunicação na placa	Tempo de processamento no hardware	Diferença de tempo (latência)
Pacote 1 MB (256 bytes)	20,44 µs	5,9 µs	1,2 µs	19,24 µs
Pacote 4 MB (1024 bytes)	29,42 µs	12,08 µs	2,74 µs	26, 68 µs
Pacote 16 MBs (4096 bytes)	60,57 µs	39,10 µs	8,88 µs	51,69 µs
Pacote 64 MBs(32768 bytes)	192,9 µs	109,69 µs	33,46 µs	159,44 µs

A partir destes ensaios se percebe que a latência de comunicação é realmente significativa, sendo particularmente relevante para pacotes de dados pequenos, quando seu atraso chega a ser mais de 16 vezes o tempo de processamento na placa (caso de um macrobloco apenas). As variações medidas entre o tempo de comunicação na placa e o tempo de processamento do hardware se explicam pelo efeito de recebimento dos dados em pacotes espaçados (Figura 112). Se a interface de comunicação permitisse um fluxo contínuo de dados na entrada da placa de hardware o ganho de desempenho seria bem mais eficiente.

A obtenção destes resultados práticos foi importante para refinar a estratégia de integração entre os módulos de software e hardware. De forma prática observa-se que a taxa de comunicação real obtida com o barramento de comunicação é bem inferior ao valor máximo teórico. Além disso, o tempo de latência da interface de comunicação é bastante significativo.

Considerando-se significativo este tempo de latência registrado nos primeiros experimentos, descartou-se a possibilidade de se realizar varredura na placa de hardware para sincronização entre as entidades (alternativa explicada na Seção 4.2.3).

Outra restrição prática se deve ao fato do *driver* de comunicação utilizado não suportar o uso de interrupções.

Sendo assim duas alternativas de integração restaram: uso de mensagens bloqueantes ou mecanismo de agendamento de tarefas.

As primeiras experiências de integração se basearam no uso de mensagens bloqueantes. Neste caso, o princípio de comunicação é o seguinte. A entidade em software requisita uma chamada da função do *driver* responsável de envio de pacote de dados para a

placa. Na prática, o *driver*, neste momento, apenas escreve os dados requisitados no espaço do *kernel* (sistema operacional), sinaliza um pedido de DMA e retorna para o aplicativo a informação de que a chamada foi concluída com sucesso. O aplicativo então requisita os dados de resposta da placa. Por se tratar de uma mensagem bloqueante o aplicativo para de executar e aguarda o retorno dos dados pedidos. Enquanto isso, o sistema operacional implementa o processo de transferência por DMA. A plataforma de hardware então recebe os dados transmitidos e inicia o processo de codificação requisitado. Ao final do processo, os dados processados pela placa são escritos no buffer de saída do *chipset* PCI, sendo então enviados para o espaço do *kernel*. O sistema operacional então sinaliza isso para o *driver*, para que este possa capturar os dados retornados e devolver isso para o aplicativo que estava bloqueado aguardando resposta. A seguir o aplicativo irá preencher suas estruturas de dados com estes valores e continuar com suas tarefas. Uma representação desta solução é ilustrada na Figura 112 na forma de diagrama de mensagens.

Este procedimento é funcional e seguro, sendo utilizado nas primeiras implementações deste projeto, porém se mostra pouco eficiente em termos de desempenho uma vez que mantém o aplicativo em software bloqueado, ou seja parado esperando por resposta.

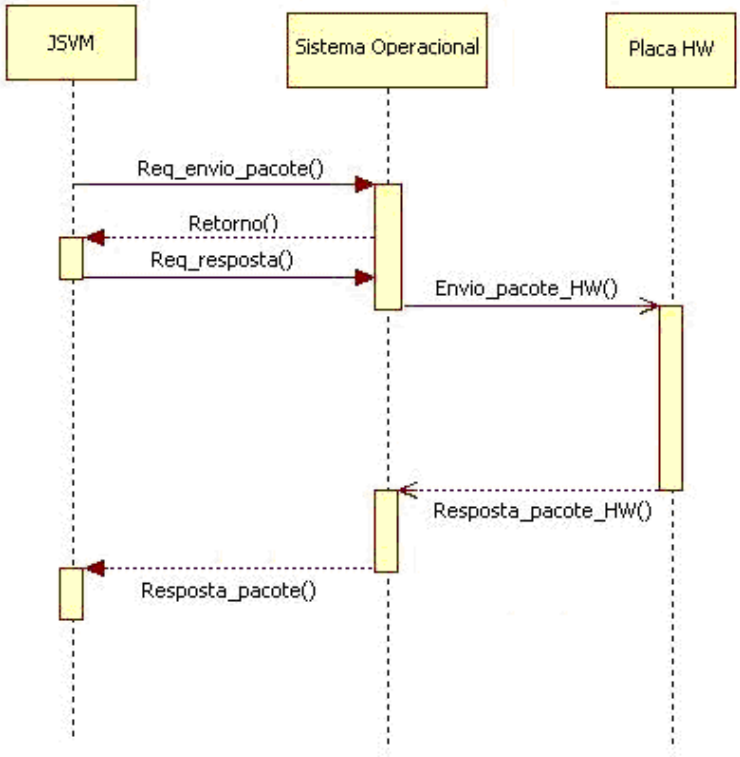


Figura 112: Diagramas de mensagens em abordagem tradicional.

Procurando aumentar o desempenho do sistema procurou-se implementar a estratégia de agendamento de serviços na placa. Para a implementação desta solução foram necessárias várias adaptações da estrutura eminentemente sequencial do software JSVM, de forma a permitir a captura e envio antecipado de dados para processamento, inclusive suportando o envio de pacotes de dados com múltiplos macroblocos, algo importante para reduzir o efeito de latência de comunicação.

Na solução implementada, cada serviço agendado (pacote de dados enviado para processamento na placa) fica armazenado em uma estrutura de dados própria que inclui, entre outras informações, o seu tempo de espera para leitura. Um módulo de software próprio é responsável por gerenciar os diversos serviços agendados. O aplicativo pode consultar através deste módulo a informação se o tempo de espera de seu serviço já esgotou ou não, ficando ao seu critério decidir a partir daí se fica esperando o tempo esgotar (mecanismo bloqueante) ou se vai executar outras tarefas e depois verificar esta condição novamente (mecanismo não bloqueante), permitindo, nesta solução, que componentes de software e hardware possam, de fato, rodar em paralelo.

Como unidade de medida para esta base de tempos, foi utilizada a informação de ticks do processador (contagem do número de ciclos de relógio da CPU). Microprocessadores x86, a partir da versão Pentium III, possuem um contador de ciclos de relógio de 64 bits que pode ser acessado para esta finalidade. Com isto, se garante uma baixa granularidade temporal e independência do sistema operacional (CHEN et al, 2009).

Esta estratégia funciona da seguinte maneira. A entidade em software requisita uma chamada da função do *driver* para envio de um pacote de dados para a placa. Neste momento, o *driver* escreve os dados solicitados no espaço do *kernel*, sinalizando um pedido de DMA feito e, a seguir, retorna ao aplicativo a informação de conclusão do serviço. O aplicativo, neste caso, não requisita, de imediato, os dados de resposta da placa, mas sim apenas registra um serviço agendado e pode a partir daí desempenhar suas funcionalidades, as quais pode ser, inclusive, o processamento dos últimos dados recebidos em um momento anterior. Cada vez que encerrou uma tarefa, ele verifica se o tempo agendado esgotou (*deadline*). Somente, neste caso, o software requisita a resposta da placa, a qual já deve ter acontecido durante o pedido em que estava trabalhando. A vantagem desta estratégia é permitir que software e hardware de fato estejam trabalhando em paralelo (Figura 113).

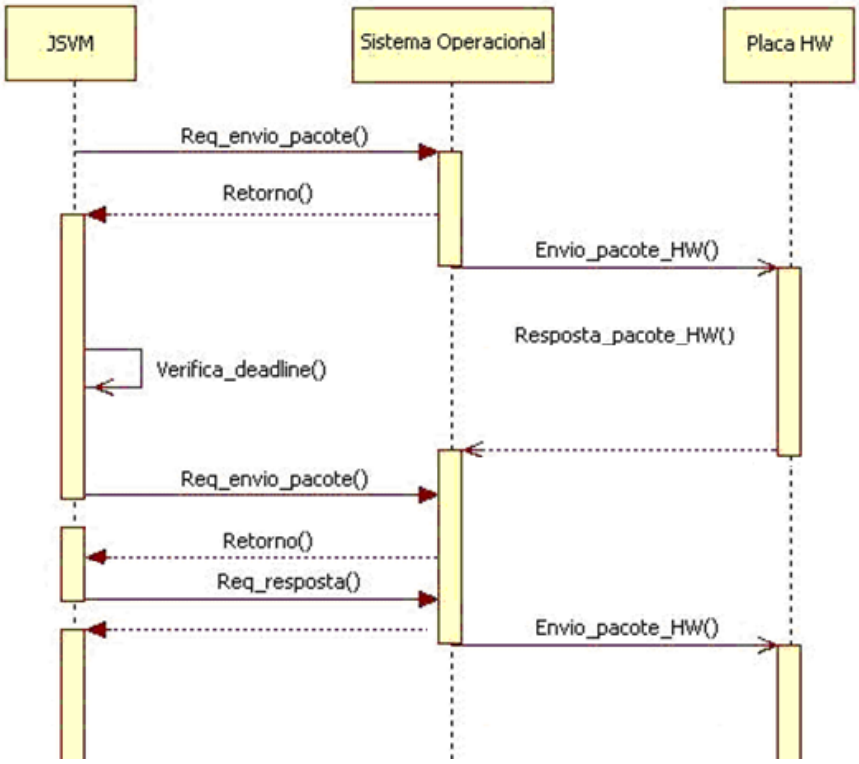


Figura 113: Diagramas de mensagens usando técnica de agendamento de serviços.

A fim de se mensurar os tempos de processamento de cada algoritmo na placa, foi implementada uma funcionalidade de verificação temporal, que pode ser anexada aos serviços enviados para a placa. Quando se requisita esta funcionalidade, o módulo em hardware desempenha as atividades de codificação requisitada e em paralelo calcula, através de um contador dedicado, o tempo total de codificação (desde a recepção do primeiro dado em sua FIFO de entrada até a escrita do último dado na FIFO de saída). Este valor de contagem é colocado na final da mensagem de resposta, juntamente com um código de verificação, que serve para confirmar que o número de contagens foi de fato atualizado.

Com base nesta funcionalidade implementada, é possível calibrar a solução para diferentes configurações (diferentes tamanhos de pacotes, mecanismos de distribuição de tarefas e mesmo computadores).

Para a implementação destes serviços, foi definido, para a aplicação, o seguinte cabeçalho para os pacotes de dados (apresentado na Figura 114). Este cabeçalho permite a sinalização do tipo de recurso exigido, a cada vez, pelo software no computador principal.

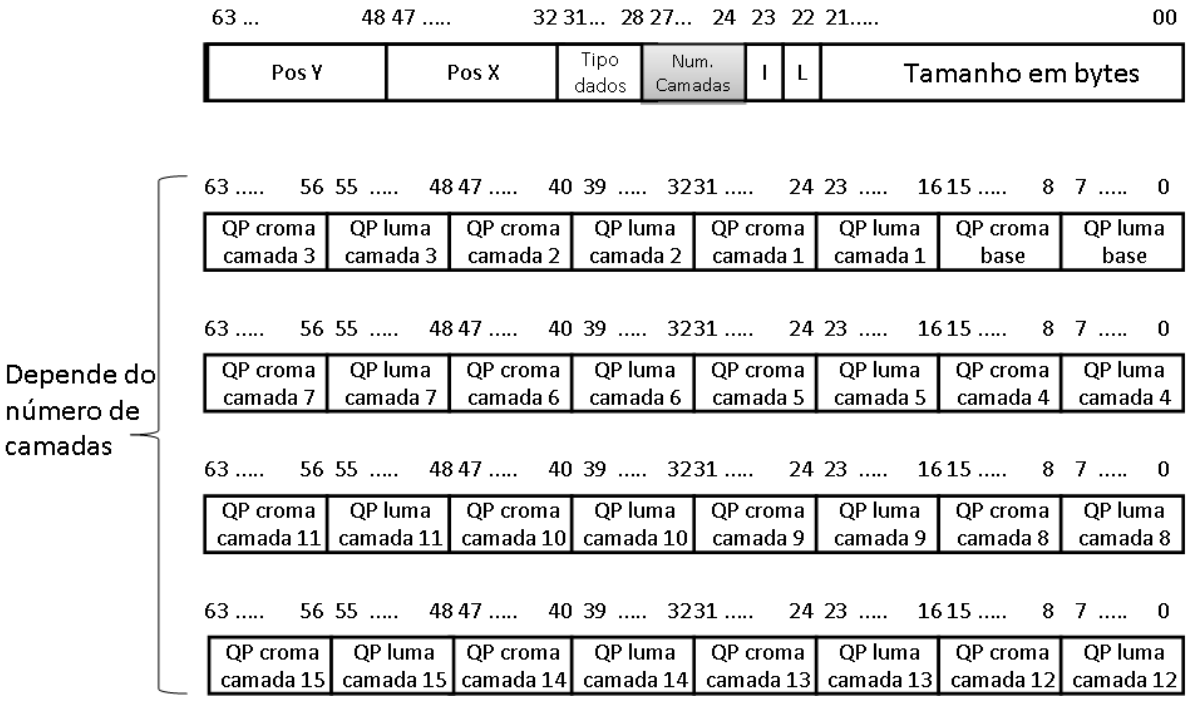


Figura 114: Cabeçalho dos pacotes enviados para a placa.

Conforme pode-se observar, o primeiro conjunto de informações dados é responsável por informar o tamanho de dados global do pacote. Logo a seguir são enviados dados de configuração conforme a seguinte descrição:

- *QP luma base*: valor do parâmetro de quantização de blocos de luminância da camada base;
- *Tipo dados*: indica o tipo de serviço requisitado (transformada, predição, entre outros);
- *QP croma base*: valor do parâmetro de quantização de blocos de crominância da camada base;
- *L*: bit que informa que os dados enviados são de luminância (=1) ou crominância (=0);
- *I*: bit que informa que os dados enviados são de intra (=1) ou inter (=0);
- *Num Camadas*: indica o número de camadas de enriquecimento requisitadas para a codificação (0 a 15);
- *Pos X*: posição na direção horizontal (x) do primeiro bloco de dados do pacote;
- *Pos Y*: posição na direção vertical (y) do primeiro bloco de dados do pacote;

- *QP luma camada n*: valor do parâmetro de quantização de blocos de luminância da camada de enriquecimento n;
- *QP croma camada n*: valor do parâmetro de quantização de blocos de crominância da camada de enriquecimento n.

6.4 RESULTADOS DOS MÓDULOS EM HARDWARE DESENVOLVIDOS

Conforme comentado anteriormente, em um primeiro momento, os módulos de hardware foram validados individualmente, visando-se a validação de seu funcionamento interno.

Para seu desenvolvimento prático, os diversos módulos deste projeto foram descritos em VHDL e sintetizados usando a ferramenta ISE versão 10.1i da empresa Xilinx (XILINX, 2010).

Cada módulo foi comparado adicionalmente com outros trabalhos relacionados da comunidade científica, para comprovação de sua validade e contribuição como arquitetura de alto desempenho.

A seguir os resultados obtidos com esta etapa são apresentados.

6.4.1 Resultados dos Módulos de Transformada

A título de referência e comparação justa com outros trabalhos, os módulos anteriormente descritos foram programados em VHDL e sintetizados para o dispositivo FPGA V2P50FF896 da família Virtex II PRO da empresa Xilinx, apesar de que, na prática, os mesmos seriam sintetizados para o dispositivo Xilinx Virtex-5 XC5VLX110T, que se encontra na plataforma de hardware XUPV5-LX110T.

A Tabela 12 apresenta, então, os resultados práticos obtidos após a síntese das arquiteturas de transformadas desenvolvidas (DCT e Hadamard) sobre o dispositivo V2P50FF896.

Tabela 12 – Resultados da arquitetura de transformadas proposta para hardware

	Elementos Lógicos (LUTs)	Período (ns)	MAmostras/s
Transformada DCT			
DCT-1D H	214	4,027	936
DCT-1D V	100	3,069	1303
DCT-2D (32 bits)	320	4,27	936
DCT-2D (64 bits)	618	4,27	1897
Transformada Hadamard			
HAD-1D H	714	9,72	412
HAD-1D V	234	4,321	925
HAD-2D	947	9,7	412
Módulo completo	1565	4,27	1897

Pode-se notar que a solução proposta atinge uma taxa de processamento de transformada realmente alta (quase 1900 MAmostras/s).

Como forma de comprovar esta afirmação, os resultados obtidos são, a seguir, comparados isoladamente com outros trabalhos relevantes de alto desempenho. Trabalhos relacionados que apresentam complexidade reduzida e baixo desempenho não foram avaliados neste etapa do trabalho, visto o foco da solução é para aumento de velocidade de codificação, tornando assim as comparações mais justas.

O trabalho de Shirani (2005) propõe uma solução de transformada DCT de alto desempenho, obtido a partir de uma combinação paralela capaz de processar uma matriz de 4x4 elementos em um único ciclo de relógio, empregando um *pipeline* de 5 estágios.

Uma abordagem similar é adotada por Dornelles (2008), que adota uma solução sem buffer de transposição e com diversas simplificações implementadas, considerando-se a simetria interna das matrizes de coeficientes.

Na Tabela 13 podem ser encontrados os resultados de síntese destes trabalhos em comparação com a solução desenvolvida de DCT.

Tabela 13 – Comparação de resultados da DCT-2D desenvolvida com outros trabalhos

	Elementos lógicos (LUTs)	Período (ns)	MAmostras/s
Shirani (2005)	644	9,3	1719
Dornelles (2008)	748	6,6	2390
DCT-2D Proposta	**623**	**4,02**	**1986**

Os resultados apontam que a solução desenvolvida apresenta um dos melhores resultados considerando-se a relação entre velocidade (período de operação) e área ocupada. Os resultados de Dornelles (2008) levam a uma taxa de dados efetiva maior, porém demandando uma taxa de entrada de 16 amostras por ciclo de relógio (ou seja, um barramento de 128 bits), o que não poderia ser atingido para a plataforma de hardware em questão. Além disso, a solução de (DORNELLES, 2008) impacta em uma maior complexidade global (cerca de 20% superior).

Para o caso das transformadas de Hadamard, a solução desenvolvida foi comparada com o trabalho de Silva (2006), que apresenta um estudo de exploração de espaço de projeto, propondo uma solução de desempenho bastante elevado (capacidade de processamento superior a 1,1 GAmostras/s).

Também foi considerado o trabalho de Agostini (2007), que propõe uma solução aprimorada para atender à demanda de codificação de vídeos HD sem elevar demais a área de chip.

A solução proposta, após ser comparada com estes trabalhos, se apresenta como a solução mais eficiente (menor complexidade e melhor desempenho).

Tabela 14 – Comparação de resultados da HAD-2D desenvolvida com outros trabalhos

	Elementos lógicos (LUTs)	Período (ns)	MAmostras/s
Silva (2006)	1140	14,04	1139
Agostini (2007)	869	4,95	202
HAD-2D Proposta	**856**	**4,44**	**1798**

Para a comparação da solução integrada os resultados obtidos consideraram dois trabalho relacionados. Agostini (2006) apresentou uma solução integrada de transformadas para visa atender à demanda de codificação de vídeos HD.

Além disso, se destaca também o trabalho de Dornelles (2008), que busca uma solução de hardware de maior desempenho, ao explorar a simetria das matrizes de coeficientes de transformada e, com isso, permitindo o processamento de uma matriz inteira de 4x4 pixels por ciclo de relógio.

A Tabela 15 apresenta a comparação de resultados considerando a solução completa de transformadas (DCT+Hadamard).

Tabela 15 – Comparação de resultados da solução de transformadas com outros trabalhos

	Elementos lógicos (LUTs)	MAmostras/s
Agostini (2006)	3140	126
Dornelles (2008)	1919	2390
Transform. Proposta	**1565**	**1801**

Assim como os resultados anteriormente obtidos com a arquitetura de transformadas DCT, a solução proposta é a que apresenta o melhor desempenho para uma arquitetura de 64 bits (Dornelles (2008) apresenta um desempenho superior, porém demandando por barramentos de 128 bits).

6.4.2 Resultados dos Módulos de Quantização

A Tabela 16 compara o resultado da arquitetura de quantização proposta em relação a outros importantes trabalhos relacionados.

Os resultados obtidos pela solução implementada apontam para um desempenho de mais de 420 MAmostras/s na sua versão básica, o que representa uma solução bastante rápida de quantização.

Em pesquisa por trabalhos similares na literatura científica, foi encontrado o trabalho de Shirani (2006), o qual propõe uma solução para o módulo de quantização ainda mais rápida (1,5GAmostras/s).

A solução de Shirani (2007), entretanto, exige barramentos maiores que 128 bits para atingir este desempenho, o que dificultaria o emprego desta solução no projeto em questão.

Além disso, cabe destacar que o trabalho de Shirani comporta apenas o processamento de dados de luminância, enquanto que o trabalho em questão suporta tanto dados de luminância como de crominância.

O trabalho de Logashanmugan (2008), por sua vez, apresenta uma solução com área bem menor, porém com uma taxa de dados bastante limitada (menor que 80 Mamostras/s)

O desempenho da solução desenvolvida, em termos da quantidade de amostras por segundo, é bem inferior ao registrado para outros módulos desta solução (transformadas DCT e Hadamard).

O principal motivo para este desempenho inferior reside no fato do módulo de quantização requerer operações de multiplicação com variáveis de 16 bits, que são computacionalmente mais extensas do que as operações de somas e deslocamentos de bits utilizadas nos módulos de transformadas, o que incrementa o período de trabalho da solução.

Tabela 16 – Comparação de resultados da solução de quantização com outros trabalhos

	Elementos lógicos (LUTs)	Período (ns)	MAmostras/s
Shirani (2007)	992	10,3	1551
Logashanmugan (2008)	546	12,54	79
Quant. Proposta	**965**	**9,45**	**423**

Neste conceito, se for considerada, a estrutura sequencial de funcionamento de um codificador intra (DCT-2D => HAD-2D => QUANT), se percebe que o módulo de quantização representa um importante gargalo para o sistema (período de 9,45ns). Por apresentar um período de trabalho maior este módulo acaba por restringir a taxa efetiva de dados da arquitetura, como pode ser visto na Figura 115.

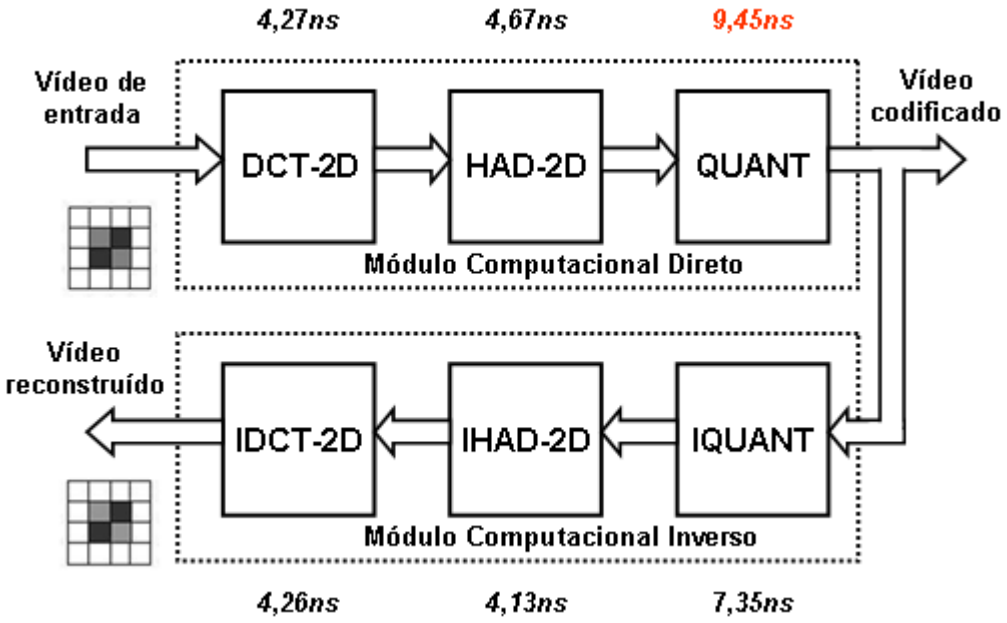


Figura 115: Arquitetura básica do módulo computacional intra.

Considerando a arquitetura seqüencial apresentada, a vazão de saída da solução seria definida pelo módulo de quantização. Procurando alternativas para esta questão foram desenvolvidas três alternativas, que são a seguir descritas:

- Módulo computacional básico;
- Módulo computacional duplicado (32 bits);
- Módulo computacional estendido (64 bits).

A seguir estas alternativas são avaliadas individualmente.

Alternativa 1 – Módulo computacional básico

Esta primeira solução usa somente uma instância de cada módulo proposto, suportando assim o processamento paralelo de até quatro amostras por ciclo de relógio, mas ficando sujeita às restrições impostas pelo módulo de quantização. Neste caso a frequência de operação ficaria limitada a 107MHz, atingindo-se uma vazão máxima de 423 MAmostras/s.

Alternativa 2 – Módulo computacional duplicado (32 bits)

A segunda abordagem avaliada (duplicado para 32 bits) procura ajustar os módulos mais lentos para atender globalmente à vazão de trabalho das demais entidades. Neste caso, os módulos mais lentos (Quant e IQuant) foram replicados, introduzindo um novo nível de

paralelismo na solução. Uma representação desta arquitetura é apresentada na Figura 116, onde se observa que o módulo mais lento ficou sendo as transformadas de Hadamard.

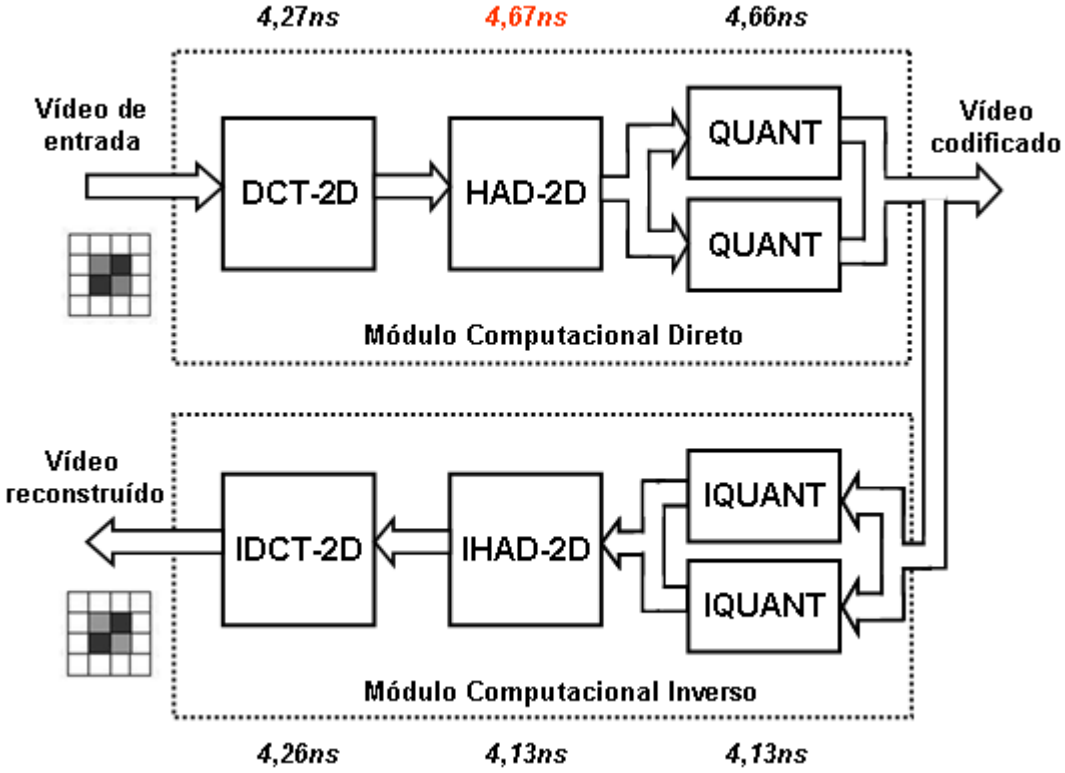


Figura 116: Arquitetura duplicada 32 bits do módulo computacional intra.

Nesta solução, os módulos replicados funcionam de forma alternada, ou seja, em um ciclo de relógio um dos módulos recebe dados novos enquanto que no ciclo de relógio seguinte é o outro módulo que recebe dados. Na prática, esta solução é implementada por um estágio de multiplexação 2x1, com sinal de controle ligado ao sinal de relógio dividido por dois. O mesmo vale no processo de tratamento das saídas destes módulos replicados.

O mesmo sinal de relógio dividido por dois é utilizado para controlar as barreiras temporais do módulos de quantização e dequantização replicados. Como ambos os módulos operam simultaneamente, porém defasados de um ciclo de relógio, a cada novo ciclo de relógio, se tem um vetor de 32 bits sem disponibilizado na saída da solução.

Com esta abordagem a freqüência de trabalho da solução pode ser duplicada. Deve-se, entretanto, observar que esta abordagem impacta no aumento da complexidade do sistema. A área ocupada aumentou em torno de 42%, porém o período de trabalho de reduziu à metade. Com isso a vazão de saída também duplicou (862 MAmostras/s).

Alternativa 3 - Módulo computacional estendido (64 bits)

A última solução desenvolvida (estendido para 64 bits) procura aumentar ainda mais o desempenho do codificador, permitindo, assim, o processamento simultâneo de até oito amostras por ciclo de relógio (conforme suportado por memórias de 64 bits). Como forma de suportar o dobro da largura de barramento esta solução implementa duas instâncias dos módulos DCT-2D e IDCT-2D e quatro instâncias de QUANT e IQUANT, operando em paralelo. Uma representação desta solução é apresentada na Figura 117, onde pode-se observar os pontos onde os barramentos de 64 bits se dividem em barramentos de 32 bits e quando são de volta unificados mantendo-se, assim, a cada etapa, a vazão de 8 amostras por ciclo de relógio.

A utilização de barramentos de 64 bits provoca a duplicação do desempenho quando comparado com a versão anterior que possuía barramentos de 32 bits, porém causando impactos ainda maiores na complexidade do sistema. Os períodos de trabalhos nesta solução são os mesmos da arquitetura anterior, porém por adotar barramentos de 64 bits de entrada, permitem o dobro da vazão de dados (1724 MAmostras/s).

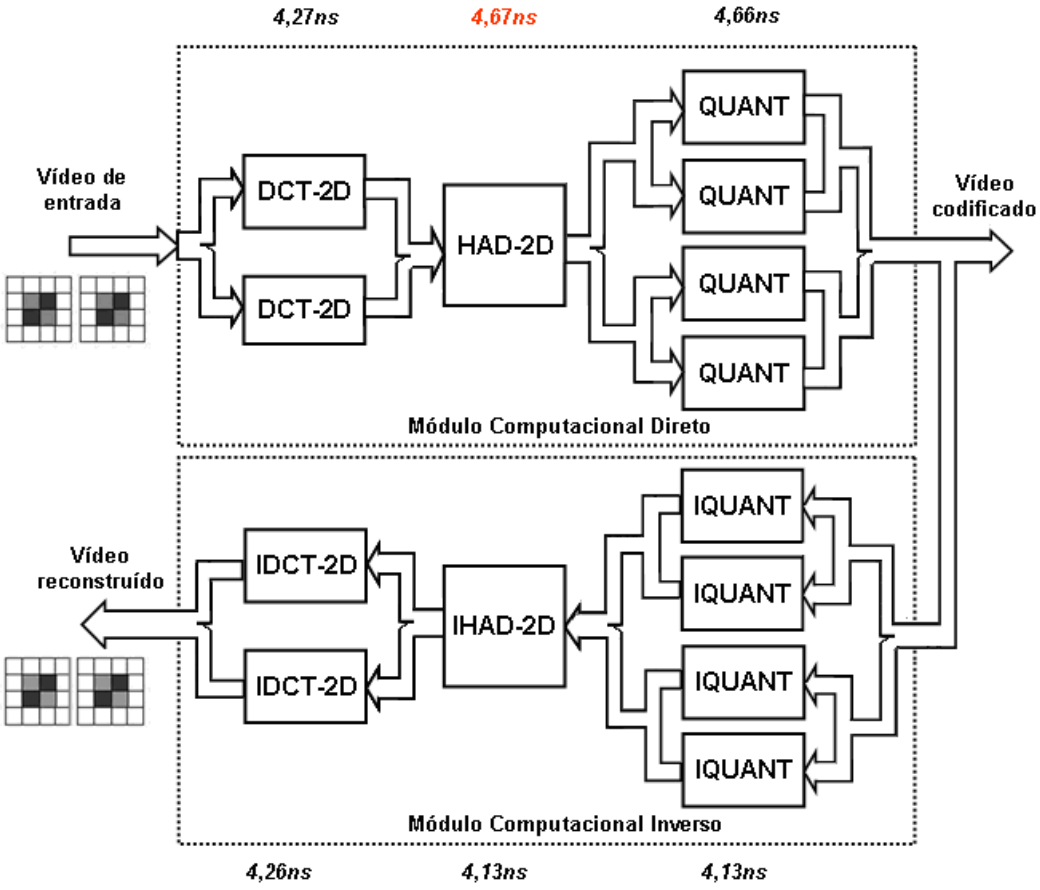


Figura 117: Arquitetura duplicada 64 bits do módulo computacional intra.

Os resultados comparativos destas três arquiteturas são apresentados na Tabela 17. Pode-se perceber que, dentre as alternativas apresentadas, a única solução capaz de suportar uma aplicação H.264/SVC 16 camadas HD é a arquitetura estendida para 64 bits.

Tabela 17 – Comparação do módulo computacional em relação a outros trabalhos

Módulo Computacional Intra Direto			
Solução	**Elementos lógicos (LUTs)**	**Período (ns)**	**MAmostras/s**
Amer (2005)	29018	14,59	2192
Korah (2008)	3924	12,09	330
Solução Básica (DCT-2D + HAD-2D + Quant)	**2126**	**9,29**	**430**
Solução Dual 32 bits (DCT-2D + HAD-2D + 2*Quant)	**3091**	**4,64**	**862**
Solução Dual 64 bits (2*DCT-2D + HAD-2D + 4*Quant)	**5343**	**4,64**	**1724**

6.4.3 Resultados da Solução Computacional Intra Completa

Como requisito para a validação da solução de computacional intra completa foi considerado, como pior caso, uma aplicação de 16 camadas de HDTV (limite da escalabilidade SNR), o que representa uma demanda teórica de 995Mpixels/s ou 1493MAmostras/s (considerando componentes de luminância e crominância em formato 4:2:0 a 30 qps).

Nesta seção, a solução proposta é verificada quanto ao atendimento a esta especificação assim como comparada com outros trabalhos relacionados.

É importante destacar que, por se tratar de um trabalho inédito, não foram localizados, até o presente momento, soluções completas especificamente voltadas para a codificação de vídeo escalável H.264/SVC em hardware.

Assim sendo, as avaliações comparativas foram feitas baseadas em trabalhos potencialmente candidatos para provimento desta aplicação.

Na prática, uma solução de codificação escalável pode ser implementada utilizando-se duas técnicas distintas: (i) criação de uma solução de elevado desempenho que possa ser utilizada para codificar a camada base e todas as demais camadas de enriquecimento em tempo de execução ou (ii) solução do tipo multi-codificador, onde um módulo de codificação H.264 convencional (codificador de camada simples) é replicado diversas vezes (tantas quantas forem as camadas desejadas).

Nesta seção, a solução proposta é comparada com soluções acadêmicas que poderiam ser adotadas em ambas as abordagens.

Inicialmente, foram buscadas soluções que atendem à primeira abordagem (solução completa de alto desempenho).

A localização de uma solução computacional intra, que integre na mesma arquitetura, as operações direta e inversa de transformadas e quantização não é trivial, visto que a maior parte dos trabalhos da comunidade, quando se busca alto desempenho, é voltada para módulos individuais.

De fato, foi localizado apenas um trabalho completo, com potencial para suportar múltiplas camadas escaláveis em formato HD, o trabalho de Dornelles (2009).

Para comparação com este trabalho, a solução proposta (seção 5.1.3) precisou ser sintetizada para a plataforma alvo Altera EP2S60 da família Stratix II.

Os resultados encontram-se apresentados na Tabela 18.

Tabela 18 – Comparação do módulo computacional em relação abordagem rápida

Solução H.264/SVC completa (abordagem rápida)			
	Elementos lógicos (LUTs)	**Periodo(ns)**	**MAmostras/s**
Dornelles (2009)	9903	10,01	578
Solução intra desenvolvida	**10658**	**4,79**	**1628**

Apesar de muito rápida a solução de Dornelles (2009) apresenta um desempenho capaz de processar apenas 7 camadas HD escaláveis. Já a solução proposta, quando aplicada para a mesma plataforma, apresenta uma área compatível (apenas 9% maior), porém com desempenho suficiente para suportar 17 camadas HD, atendendo, assim, as especificações máximas requeridas por esta aplicação segundo a norma H.264/SVC.

Na tabela a seguir, a solução completa proposta é comparada com outros trabalhos relacionados, porém, neste caso, buscando-se uma abordagem multi-codificador. Para tanto, foram selecionadas soluções completas de módulos computacionais intra especialmente otimizadas para trabalhar com codificação de vídeo HD.

Os três trabalhos relacionados utilizam um dispositivo XC2VP30 FPGA como plataforma alvo. Suas características, quando comparadas com a solução atual, são resumidas na Tabela 19.

Tabela 19 – Resultados do codificador intra em relação a trabalhos para uma camada

Cenário	**Módulos Computacionais Intra Diretos**			
	Solução	**Elementos lógicos (LUTs)**	**Período (ns)**	**MAmostras/s**
A	Logashanmugam (2008)	546	12,54	79
B	Korah (2008)	3924	12,09	330
	Solução desenvolvida	**5343**	**4,64**	**1724**
Cenário	***Módulos Computacionais Intra Inversos***			
	Solução	**LUTs**	**Período (ns)**	**MAmostras/s**
C	Agostini (2007)	3249	7,54	132
	Solução desenvolvida	**3359**	**4,64**	**1724**

Na comparação com trabalhos de computação direta, a solução desenvolvida é a que ocupa mais área, porém apresenta uma taxa de dados compatível com as altas demandas de um codificador SVC.

Já na comparação com o módulo computacional inverso selecionado, a solução proposta apresenta um consumo de área compatível, porém um desempenho bem superior (1724 MAmostras/s). Estes resultados preliminares se justificam, pois os demais trabalhos relacionados (cenários A, B e C) foram inicialmente desenvolvidos para operação não escalável. Estes módulos podem, entretanto, ser replicados para produzirem uma abordagem multi-codificador, de forma que sejam, realmente capazes de suportar uma aplicação escalável.

Com base nas demandas consideradas da aplicação alvo, considera-se assim a necessidade de que estes módulos de camada simples precisam ser replicados 16 vezes (número de camadas), o que é apresentado na Tabela 20.

Tabela 20 – Comparação do módulo computacional escalável com outros trabalhos

Cenário	**Módulos Computacionais Intra Diretos**			
	Solução	**Elementos lógicos (LUTs)**	**Período (ns)**	**MAmostras/s**
A (x16)	Logashanmugam (2008)	8736	12,54	1264
B (x16)	Korah (2008)	62784	12,09	5280
Cenário	**Módulos Computacionais Intra Inversos**			
	Solução	**Elementos lógicos (LUTs)**	**Período (ns)**	**MAmostras/s**
C (x16)	Agostini (2007)	51984	7,54	2112
Cenário	**Módulos Computacionais Escaláveis Intra Inversos**			
	Solução	**Elementos lógicos (LUTs)**	**Período (ns)**	**MAmostras/s**
B+C (x16)	Solução Multi-codificador	114768	7,54	2112
	Solução intra completa desenvolvida	**11485**	**4,64**	**1724**

A solução apresentada no cenário A (Tabela 19), mesmo quando replicada 16 vezes (Tabela 20) ainda não suporta a demanda de 1493 MAmostras/s.

Considerando isso os cenários B e C foram usados para compor uma arquitetura computacional escalável.

A solução referenciada como cenário “B+C (x16)” suporta mais de 2 GAmostras/s, garantindo assim as demandas SVC. Entretanto, a área total ocupada para esta solução é bastante elevada.

Comparada com a solução desenvolvida, que adota a solução iterativa apresentada na Seção 5.1.3, que integra os módulos computacionais diversos e inversos, memórias MB e gerenciador de camadas SVC (Figura 82), pode-se observar a redução significativa de área (cerca de 10 vezes menor), ainda suportando a taxa máxima requerida para a aplicação escalável de vídeos HD.

6.4.4 Resultados dos Módulos de Filtro Anti-blocagem

A arquitetura proposta de filtro anti-blocagem foi implementada em VHDL e sintetizada em placa para fins de validação do correto comportamento da implementação. Sua validação quando foi feita comparando seus resultados com o software de referência JSVM para vídeos conhecidos como Foreman e Crew.

Conforme descrito na Seção 5.2 foram desenvolvidas duas soluções distintas de filtro anti-blocagem:

- Solução básica de 32 bits: recebe, a cada ciclo de relógio, quatro amostras. Neste caso, a vazão de dados da solução garante entrada e saída de uma linha ou coluna de dados por unidade de relógio.
- Solução estendida de 64 bits: recebendo oito amostras por ciclo de relógio (duas linhas ou colunas simultâneas). Utiliza uma arquitetura de *pipeline* mais longo para permitir uma vazão de dados.

A seguir, as duas arquiteturas são comparadas, quando sintetizadas para uma mesma arquitetura alvo (dispositivo Virtex5 LX110T). Observa-se que a versão básica ocupa mais do que a metade da área da versão estendida, o que já era esperado, porém com metade da taxa de dados efetiva.

Para comparação com outros trabalhos relacionados, a arquitetura desenvolvida foi sintetizada para um dispositivo ASIC de tecnologia 180µm, utilizando-se a ferramenta RTL Compiler v06.20-p003_1 da empresa Cadence.

A Tabela 21 traz os resultados de implementação, obtidos com esta síntese, juntamente com os resultados de outras arquiteturas de filtragem citadas na Seção 5.2. Todas as arquiteturas propostas são implementadas para suportar um fluxo contínuo de 32 bits atualizadas por ciclo de relógio.

Observando-se a tabela, percebe-se que a solução básica é a que detém a menor ocupação de células lógicas. Isto se deve basicamente ao fato de se adotar uma abordagem que implementa o reuso de operações internas, conforme apresentado na descrição da solução desenvolvida. Já a solução estendida, que ocupa uma área maior é a que apresenta o melhor desempenho (menor número de ciclos de relógio por macrobloco de todas as soluções avaliadas).

Tabela 21 – Comparação do módulo de filtragem em relação a outros trabalhos

	Hu (2007)	Chen (2007)	Wang (2010)	Lai (2010)	**Proposta Básica**	**Proposta Estendida**
Número ciclos/MB	192	160	200	212	**192**	**132**
SRAM	3 x 160x32	264x32	Nx32	72x32	**Nx32**	**(N/2)x64**
Buffers transposição	2	8	2	2	**2**	**4**
Quant. de portas (gates)	14,8K	20,8K	37,3K	12,2K	**12,2 K**	**20,2 K**

Após a validação funcional dos módulos desenvolvidos em ambiente de teste estes módulos de filtragem foram implementados na plataforma de hardware escolhida para fins de comparação de desempenho obtido.

A seguir, as duas arquiteturas são comparadas, quando sintetizadas para uma mesma arquitetura alvo (dispositivo Virtex5 LX110T). Observa-se que a versão básica ocupa mais do que a metade da área da versão estendida, o que já era esperado, porém com metade da taxa de dados efetiva. A versão estendida é a única que consegue atingir uma vazão suficiente para suportar o processamento de 16 camadas HD escaláveis.

Tabela 22 – Comparação entre arquiteturas de filtro anti-blocagem

Utilização de recursos	Básica	Estendida
Número de Slices	995	1754
Número de Slice Flip Flops	1534	2782
Número de LUTs	254	586
Período de trabalho	3,76	4,67
Taxa de dados máxima (MAmostras/s)	1063	1713

6.4.5 Módulo de Predição entre Camadas

A versão proposta de predição entre camadas foi especialmente desenvolvida para aprimorar seu desempenho, baseando-se em uma abordagem de busca zonal com diamante pequeno, utilizando-se estruturas de dados subamostradas e especialmente alinhadas com os barramentos de memória.

Um parâmetro importante para avaliação de desempenho de um algoritmo de predição é o parâmetro Ciclos/MV, que representa a relação entre o número total de ciclos usados e o número de vetores de movimento calculados para uma sequência de vídeo inteira.

Este parâmetro não é fixo, pois depende do tipo de movimentação encontrada, tamanho da janela de busca, tamanho dos macroblocos escolhidos, entre outras configurações.

Para a geração destes dados de forma prática, o algoritmo desenvolvido foi aplicado para diferentes vídeos, registrando-se o tempo interno de codificação do algoritmo. No caso, para que este experimento fosse possível, foi criado um contador que inicia a cada recepção de um quadro completo de vídeo, sendo incrementado a cada ciclo de relógio.

Quando o quadro inteiro de vídeo for processado o valor deste contador é enviado anexado à mensagem de resposta. A determinação do parâmetro é então obtido, dividindo-se esta contagem pelo número de macroblocos presentes no vídeo alvo.

A seguir se apresentam os resultados obtidos para diferentes vídeos. No caso, para estes experimentos, a janela de busca estava ajustada para um tamanho de 96x96, com tamanho de macrobloco fixo em 16x16 pixels.

Tabela 23 – Comparação entre arquiteturas de filtro anti-blocagem

Vídeos (Formato CIF@30qps)	Valor do contador	Ciclos/MV calculado
City	18948	48
Crew	36832	93
Harbour	10870	27
Foreman	17791	45

Conforme esperado este valor oscilou bastante, desde 27 para o vídeo Harbour até 93 ciclos para o vídeo Crew (uma diferença de indica que o vídeo Crew, em média, leva cerca de quatro vezes mais tempo para ser codificado do que o vídeo Harbour, por exemplo). A

explicação para esta movimentação se deve aos padrões de movimentação variados das sequencias de vídeo.

Avaliando-se o comportamento do algoritmo no pior caso (93 ciclos/MV) se percebeu que no caso do vídeo Crew, o algoritmo de diamante pequeno passava por várias etapas de movimentação ao longo do ponto inicial, muitas vezes entrando em laços de repetição involuntários. Foi utilizado como proteção, para estes casos, o limite máximo de cinco iterações. Este valor for escolhido a partir de experimentos em laboratório. Com limites menores que este a perda de qualidade passava a ficar significativa (acima de 0,3dB) e para valores maiores o tempo de codificação aumentava, porém sem ganhos de qualidades significativos (cerca de 0,2dB).

A seguir a solução proposta é comparada com outras implementações relevantes de algoritmos de estimativa rápidos. A comparação apresentada considera questões práticas como consumo de elementos lógicos e memória.

Todas as soluções avaliadas nesta tabela foram sintetizadas para um dispositivo Virtex 4 XC4LX25.

Tabela 24 – Comparação do módulo de predição em relação a outros trabalhos

Algoritmo	**Resultados**		
	LUTs	**BRAM**	**Ciclos/MV**
Agostini (2007)	37561	42	333
Sayed (2008)	22010	40	596
Porto (2008)	3610	32	140
Ndili (2010)	14600	*	300
Solução Proposta	**11705**	**82**	**93 (pior caso)**

* Não informado pelo autor.

Mesmo considerando-se o elevado nível de paralelismo adotado pelo modulo de cálculo de SAD (cinco estruturas são usadas para implementar o calculo completo do padrão de diamante pequeno), a complexidade global da solução desenvolvida não é tão grande quando comparada com outros trabalhos. Na prática, somente um trabalho ocupa uma quantidade bem menor de elementos lógicos (PORTO, 2008).

Por outro lado, a presente arquitetura representa a solução mais rápida em termos de ciclos por vetor de movimento, um parâmetro muito importante quando se busca uma solução tempo real, principalmente para o caso de uma aplicação de codificação de vídeo escalável H.264/SVC.

6.5 RESULTADOS PRÁTICOS DO CODIFICADOR INTRA

Após a avaliação isolada destes diferentes módulos, de forma isolada, foi realizada a integração destes módulos de hardware com o software de referência JSVM. A solução integrada em placa inclui o módulo computacional intra completo, contendo os módulos diretos e inversos de transformadas, quantização e filtro anti-blocagem.

A estratégia adotada para integração foi alterar as funções em software, onde os cálculos de codificação intra eram processados, para que fossem feitas chamadas aos módulos em hardware. Estas chamadas se caracterizavam pela montagem de pacotes de dados que eram transmitidos via DMA para a placa via um *driver* PCI Express. Uma vez que os dados tenham sido processados pela placa o software pode buscá-los, também através de uma chamada do *driver* PCI Express, e passá-los para as funções de entropia. A fim de se evitar o travamento do software durante o tempo em que o hardware estiver processando as informações passadas, adotou-se uma estratégia de escrita em blocos antecipada.

Nesta estratégia, a cada vez que um novo conjunto de dados é buscado da placa se prepara, por antecipação, o conjunto subsequente a ser processado pelo software. Assim enquanto os dados atuais estão sendo processados pelas funções de entropias, novos dados estão sendo processados em paralelo pela placa em hardware. Os ganhos obtidos com a paralelização destas operações dependem, obviamente, da quantidade de dados enviados por pacote. Assim, foram feitos vários experimentos considerando diferentes quantidades de macroblocos por mensagem. Limitações do *driver* impediram os ensaios de pacotes maiores que 32768bytes, o que representa um limite de no máximo 128 macroblocos de 16x16 pixels por mensagem. Os números de macroblocos alocados para cada pacote foram escolhidos de forma a que cada mensagem contenha uma parte inteira do total de macroblocos de um quadro.

Para estes experimentos, foram utilizados vários vídeos de referência agrupados em distintas resoluções de forma a expor o algoritmo a condições diversas. Na prática, foram selecionados três formatos de vídeos distintos (CIF, 4CIF e HD).

Com base nestes ensaios, é possível avaliar de forma isolada os efeitos de diferentes padrões de movimentação (obtido variando-se os vídeos de entrada), bem como o efeito de mudança de tamanho da imagem (comparando-se, por exemplo, o mesmo vídeo nos formatos CIF e 4CIF).

Basicamente, os vídeos escolhidos foram:

- Formato CIF e 4CIF
    - Foreman: Vídeo onde um personagem conversa com a câmera e aponta, em certo momento, para uma cena de construção ao lado, quando, então, a imagem se desloca nesta direção;
    - City: Visão panorâmica filmada de um helicóptero enquanto dá uma volta sobre a torre do prédio Empire State;
    - Crew: Tripulação de um ônibus espacial filmada na plataforma enquanto se dirigem para a aeronave;
    - Harbour: Filmagem de um cais, onde se observam barcos se deslocando na saída e chegada.

- Formato HD
    - Pedestrian: Cena urbana de uma esquina movimentada, onde se observam pedestres e ciclistas se deslocando em diferentes direções e com diferentes velocidades;
    - RushHour: Cena de uma avenida urbana movimentada, onde se observam os movimentos de veículos e passageiros em sentidos variados.

Na prática os experimentos realizados não mostraram variações significativas entre vídeos de mesmo formato (oscilações menores de 0,5%).

Isto, de fato, já era esperado, pois o desempenho de um codificador intra basicamente não é afetado pelo formato da imagem (a qualidade final sim, porém não o tempo de processamento).

Sendo assim os resultados obtidos foram apresentados considerando-se as médias dos diferentes vídeos. Na Tabela 25 se apresentam os resultados do processamento de luminância.

Para comparação entre valores, todos os tempos apresentados são apresentados na unidade de *ticks* (ciclo de relógio da CPU), por se tratar de uma unidade de medida genérica, ou seja, que independe da frequencia de operação do computador.

Tabela 25 – Resultados de codificação de luminância obtidos para diferentes vídeos

		Versão em software			Versão integrada com hardware				Ganho relativo (%)
Formato	Número de Macrolocos	Tempo por bloco	Total para conjunto	Total por quadro	Tempo inicial	Tempo por bloco	Total para conjunto	Total por quadro	
CIF	1	14600	14600	5781600	29800	29800	29800	11800800	48,993
	2	14500	29000	5781600	15600	3700	19300	3821400	151,295
	3	14500	43500	5781600	16600	2400	21400	2824800	204,672
	6	14500	87000	5781600	18700	1400	25700	1696200	340,856
	9	14500	130500	5781600	21200	1200	30800	1355200	426,623
	11	14500	159500	5781600	23300	1100	34300	1234800	468,221
	22	14500	319000	5781600	34200	920	53520	963360	600,149
	33	14500	478500	5781600	52500	920	81940	983280	587,991
	44	14500	638000	5781600	72000	870	109410	984690	587,149
	66	14500	957000	5781600	128000	860	183900	1103400	523,980
	99	14600	1445400	5781600	245000	850	328300	1313200	440,268
4CIF	1	14500	14500	22968000	36300	36300	36300	57499200	39,945
	2	14500	29000	22968000	15500	3800	19300	15285600	150,259
	3	14500	43500	22968000	15800	2300	20400	10771200	213,235
	6	14500	87000	22968000	17800	1400	24800	6547200	350,806
	9	14500	130500	22968000	19000	1100	27800	4892800	469,424
	11	14500	159500	22968000	21000	1000	31000	4464000	514,516
	22	14500	319000	22968000	27300	1000	48300	3477600	660,455
	33	14500	478500	22968000	36500	900	65300	3134400	732,772
	44	14500	638000	22968000	45800	800	80200	2887200	795,511
	66	14500	957000	22968000	70200	800	122200	2932800	783,142
	99	14500	1435500	22968000	119000	800	197400	3158400	727,204
HD	1	14600	14500	118320000	28829	28829	28829	235244640	50,297
	2	14500	29000	118320000	15500	3500	19000	77520000	152,631
	3	14500	43500	118320000	15700	2300	20300	55216000	214,286
	6	14500	87000	118320000	17600	1300	24100	32776000	360,996
	9	14500	130500	118320000	19600	1100	28400	25749333	459,507
	18	14500	261000	118320000	24500	1000	41500	18813333	628,916
	24	14500	348000	118320000	27300	930	48690	16554600	714,726
	30	14500	435000	118320000	31000	930	57970	15767840	750,388
	45	14500	652500	118320000	40200	850	77600	14071466	840,851
	60	14500	870000	118320000	46500	850	96650	13144400	900,155
	100	14600	1450000	118320000	70200	800	149400	12191040	970,549

A análise dos resultados aponta para ganhos maiores, conforme o número de dados por pacote aumenta. Isso se deve basicamente pelo fato de se reduzir a representatividade do tempo de latência do driver PCI Express (atraso relacionado ao sistema operacional).

Entretanto, para pacotes muito grandes, a eficiência acaba sendo prejudicada pela rotina de inicialização de cada quadro, momento em que os buffers de trabalho do software de referência são reinicializados pelo software de referência, situação que ocorre uma vez a cada quadro.

Nestes momentos, a escrita antecipada não é possível, pois os blocos subseqüentes ainda não estão disponíveis para uso. Esta medida está representada na tabela como “tempo inicial” dos pacotes de dados na versão em hardware.

O resultado final de ganho de desempenho é apresentado de forma gráfica na Figura 118, onde pode-se identificar um ganho de até 970% (solução integrando hardware é 9,7x mais rápida que uma solução de software puramente) para pacotes de 100 macroblocos em sequências de vídeo HD, durante um processamento de luminância.

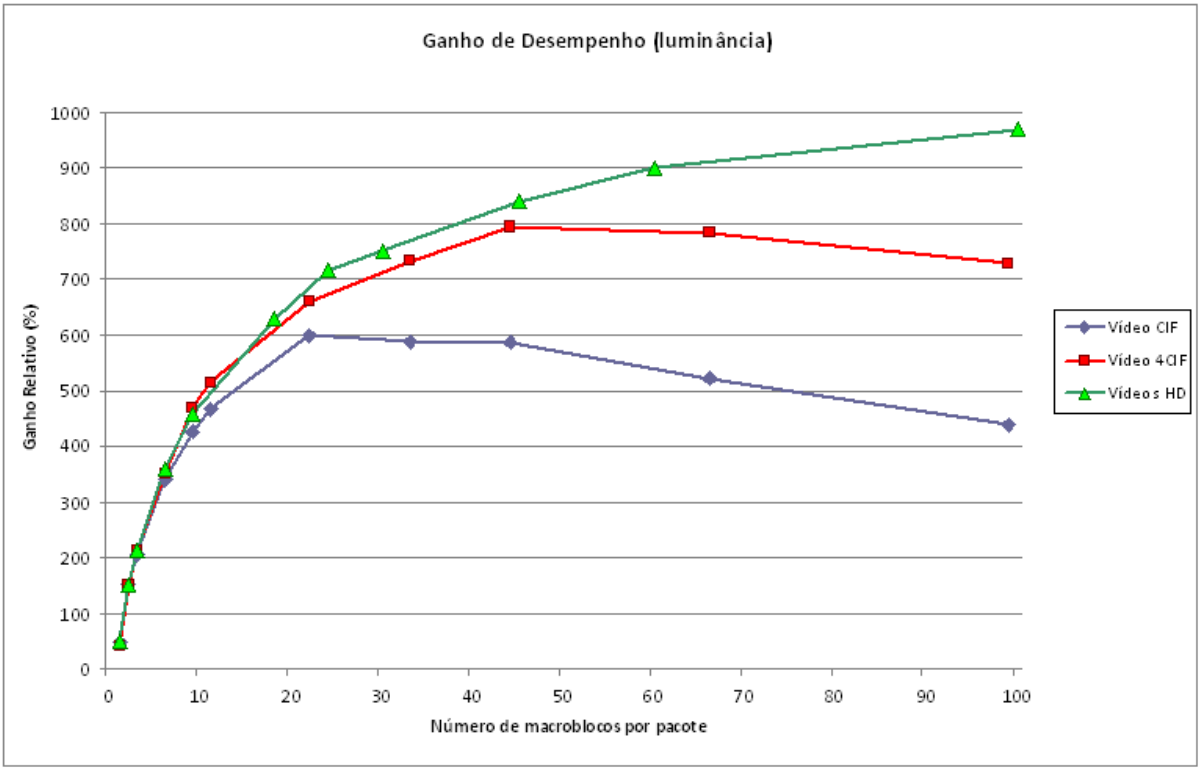


Figura 118: Ganhos de desempenho da solução intra para processamento de luminância

Na Tabela 26, por sua vez, se apresentam os mesmos ensaios considerando as amostras de crominância.

Tabela 26 – Resultados de codificação de crominância obtidos para diferentes vídeos

		Versão em software			Versão integrada com hardware				
Formato	Número de Macrolocos	Tempo por bloco	Total para conjunto	Total por quadro	Tempo inicial	Tempo por bloco	Total para conjunto	Total por quadro	Ganho relativo (%)
CIF	2	8600	17200	3524400	15500	3800	19300	3821400	**92,228**
	4	8600	34400	3524400	16800	2000	22800	2257200	**156,140**
	6	8600	51600	3524400	17600	1400	24600	1623600	**217,073**
	12	8600	103200	3524400	21200	1100	33300	1098900	**320,720**
	18	8600	154800	3524400	25800	1000	42800	941600	**374,299**
	22	8600	189200	3524400	28800	900	47700	858600	**410,482**
	44	8600	378400	3524400	47800	900	86500	784800	**449,083**
	66	8600	567600	3524400	91600	800	143600	804600	**438,031**
	88	8600	756800	3524400	130000	800	199600	836190	**421,483**
	132	8600	1135200	3524400	273000	800	377800	4533600	**355,214**
	198	8600	1762200	3524400	446000	830	609510	1219020	**289,117**
4CIF	2	8600	17200	13622400	14800	3700	18500	14652000	**92,973**
	4	8600	34400	13622400	16300	1800	21700	8593200	**158,525**
	6	8600	51600	13622400	17500	1500	25000	6600000	**206,4**
	12	8600	103200	13622400	19700	1100	31800	4197600	**324,528**
	18	8600	154800	13622400	22200	1000	39200	3449600	**394,898**
	22	8600	189200	13622400	24100	1000	45100	3247200	**419,512**
	44	8600	378400	13622400	33600	900	72300	2602800	**523,374**
	66	8600	567600	13622400	48700	800	100700	2416800	**552,409**
	88	8600	756800	13622400	65600	800	135200	2433600	**558,936**
	132	8600	1135200	13622400	108200	800	213000	2556000	**532,958**
	198	8600	1762200	13622400	194400	800	352000	2816000	**483,75**
HD	2	8600	17200	70176000	15200	3600	18800	76704000	**91,489**
	4	8600	34400	70176000	16300	1800	21700	44268000	**158,525**
	6	8600	51600	70176000	16800	1300	23300	31688000	**221,459**
	12	8600	103200	70176000	19400	1000	30400	20672000	**339,474**
	18	8600	154800	70176000	21900	980	38560	17480533	**401,452**
	36	8600	309600	70176000	28200	880	59000	13373333	**524,746**
	48	8600	412800	70176000	31500	850	71450	12146500	**577,747**
	60	8600	516000	70176000	35600	830	84570	11501520	**610,145**
	90	8600	774000	70176000	47800	800	119000	10789333	**650,420**
	120	8600	1032000	70176000	59200	800	154400	10499200	**668,394**
	200	8600	1720000	70176000	97500	790	254710	10392168	**675,278**

Neste caso, o resultado final de ganho de desempenho chegou a 675% (solução em hardware 6,7x mais rápida que a de software para pacotes de 200 macroblocos de crominância) em sequências HD (Figura 119).

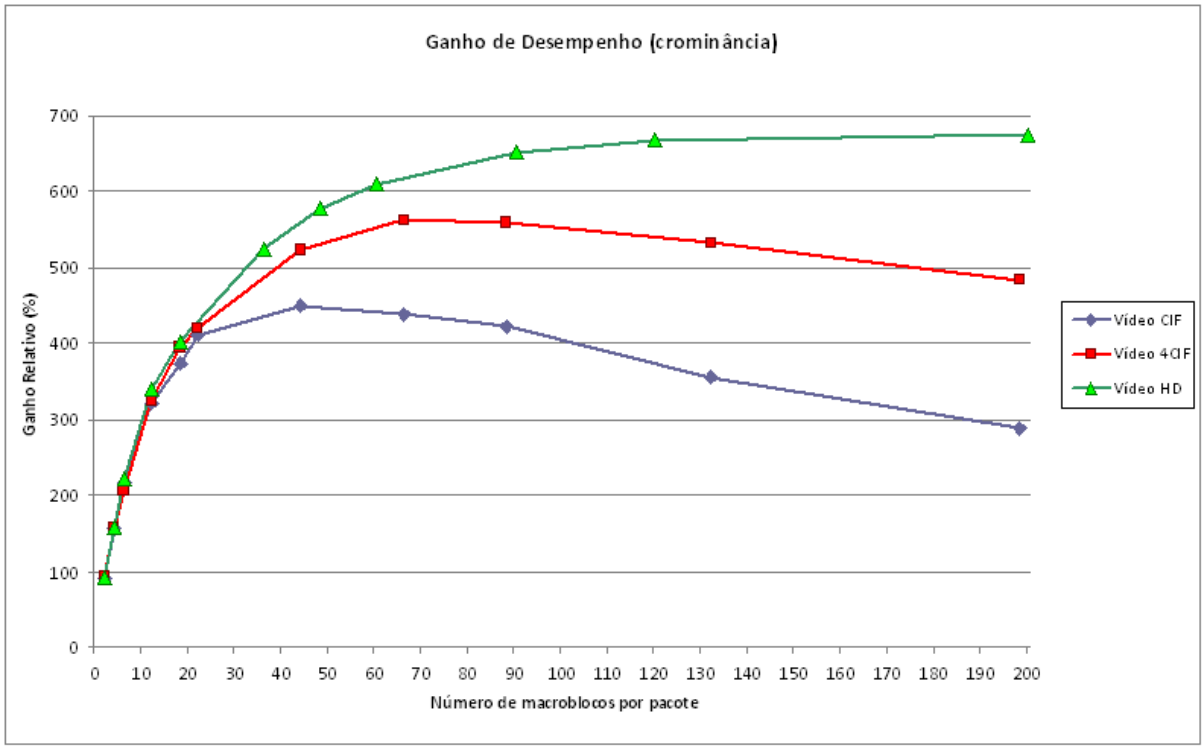


Figura 119: Ganhos de desempenho da solução intra para processamento de crominância

6.6 RESULTADOS PRÁTICOS DO CODIFICADOR INTER

A solução de codificador inter implementada em placa foi também validada e comparada com os resultados do software de referência JSVM. Resultados práticos registraram ganhos na ordem de 600% (6 vezes mais rápidos), quando comparados com vídeos como Harbour e Crew 4 CIF @30fps.

Estes ganhos podem ser considerados bastante significativos uma vez que foram comparados com implementações também rápidas de estimativa de movimento.

As tabelas a seguir apresentam os resultados da solução proposta em comparação com a aplicação tradicional (codificação implementada inteiramente em software pelo JSVM) para diferentes vídeos e diferentes resoluções de tela.

Os parâmetros adotados para ambos os casos foram: desabilitação de particionamento 16x8 e 8x8, desabilitação de refinamento de quarto de pixel, um quadro de referência apenas e janela de busca de tamanho 32x32.

Os resultados para CIF e 4CIF são apresentados em pontos distintos da tabela, pois as taxas de saída (*bitrate*) são diferentes.

Tabela 27 – Comparação de algoritmos de predição do JSVM em termos de PSNR (dB)

Formato	Vídeo	Algoritmo	PSNR (dB)			
			1000 kbps	1500 kbps	2000 kbps	2500 kbps
CIF	City	Normal	34,7678	37,8233	40,5089	43,1157
		Proposto	34,7849	37,8241	40,5099	43,1169
	Crew	Normal	38,5163	41,6601	44,2961	46,6752
		Proposto	38,5090	41,6555	44,2983	46,6661
	Harbour	Normal	45,4218	48,5536	48,5536	48,5536
		Proposto	45,2052	48,5831	48,5977	48,5977
	Foreman	Normal	30,7284	33,9823	36,8669	39,6029
		Proposto	30,7178	33,9785	36,8518	39,6089
Formato	Vídeo	Algoritmo	PSNR (dB)			
			2000 kbps	3000 kbps	4000 kbps	5000 kbps
4CIF	City	Normal	32,2103	34,4162	36,2314	37,8489
		Proposto	32,1949	34,5142	36,2861	37,8510
	Crew	Normal	36,9543	39,1244	40,8464	42,1874
		Proposto	36,9358	39,0451	40,6177	42,1951
	Harbour	Normal	29,4979	31,8918	34,0313	35,7693
		Proposto	29,5292	31,8744	34,0012	35,8187

Tabela 28 – Comparação do tempo de execução (s) dos algoritmos de predição do JSVM

Formato	Vídeo	Algoritmo	Tempo de execução (s)			
			2000 kbps	3000 kbps	4000 kbps	5000 kbps
CIF	Crew	Normal	5010,9350	5197,6050	5177,6050	5227,8050
		Proposto	305,5721	303,5438	301,281	300,1815
CIF	City	Normal	5060,3171	5157,672	5201,5096	5327,2342
		Proposto	304,522	302,128	301,311	300,932
CIF	Harbour	Normal	4927,4548	4991,785	5012,9532	5103,345
		Proposto	303,2451	306,258	310,522	305,754
4CIF	Crew	Normal	1240,6533	1276,4456	1284,232	1302,4304
		Proposto	75,338	73,405	72,417	73,013
4 CIF	City	Normal	1230,337	1226,873	1284,373	1302,924
		Proposto	74,343	72,357	72,038	71,399
4 CIF	Harbour	Normal	1202,633	1206,355	1210,388	1196,591
		Proposto	73,341	72,034	71,939	72,011

As avaliações de perda de qualidade pelo algoritmo adotado de predição podem ser consideradas como bastante reduzidas (em média menores que 0,1dB), conforme pode-se observar nos gráficos a seguir (Figura 120 e 121), onde os resultados são colocados lado a lado, sem percepção visual de diferença.

Figura 120: Comparação de qualidade para vídeo City 4cif

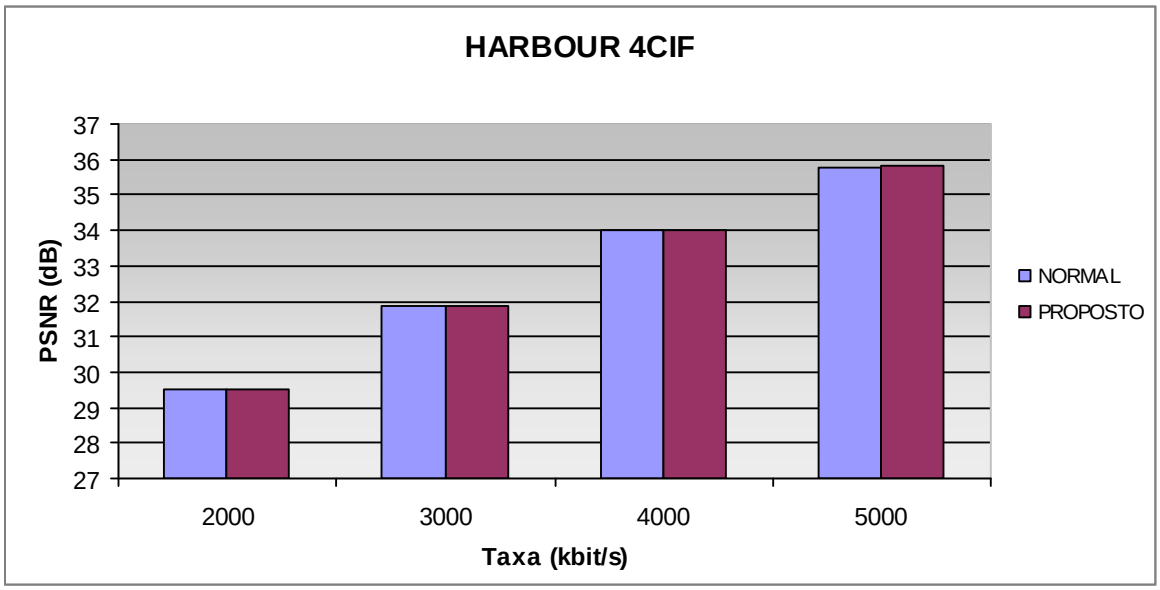


Figura 121: Comparação de qualidade para vídeo Harbour 4cif

Já o ganho de desempenho registrado é bastante significativo para todos os vídeos analisados.

Para ilustrar isso, nas Figuras 122 e 123 são apresentados os tempos médios de codificação para os dois casos (normal e proposto).

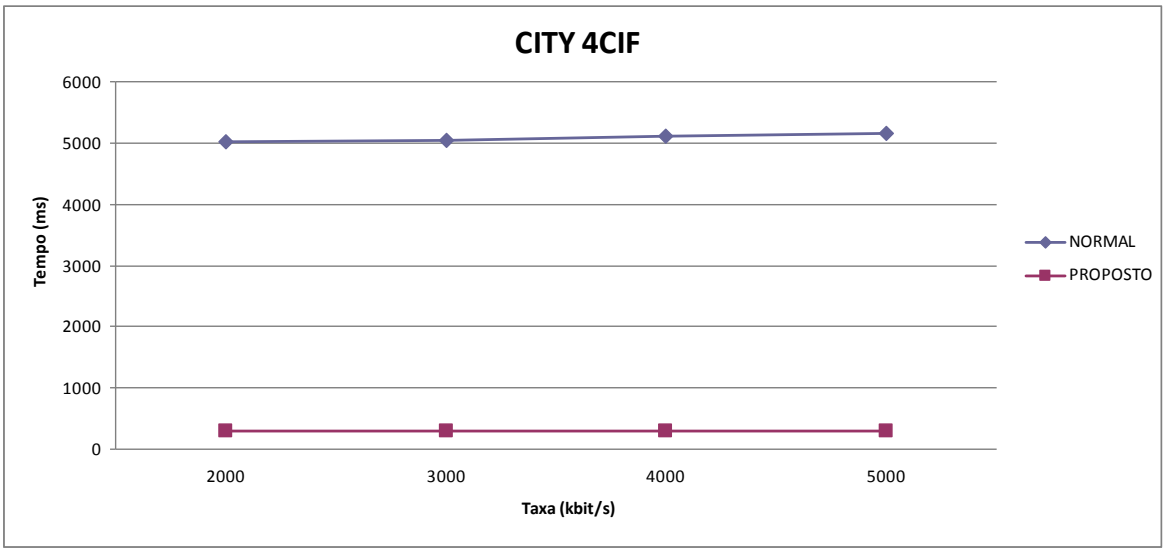


Figura 122: Comparação do tempo de execução (ganho de desempenho) para City 4cif

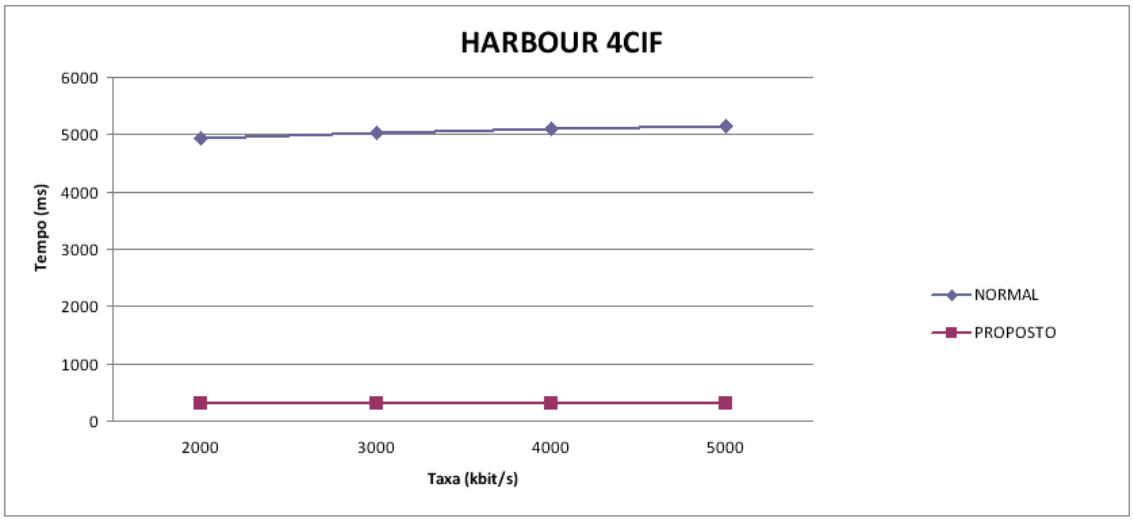


Figura 123: Comparação do tempo de execução (ganho de desempenho) para Harbour 4cif

No melhor caso, foi registrado um ganho na ordem de 1680% (solução proposta é 16,8x mais rápida que uma solução de JSVM convencional).

De forma complementar, para verificação visual de funcionamento, foi utilizada a ferramenta ChipScope Pro Analyzer tool que permite monitorar o comportamento interno de uma implementação.

A Figura 124 ilustra uma tela capturada desta solução durante operação, quando a mesma foi sintetizada na placa de desenvolvimento XUP5V110T.

Pode-se observar na figura capturada que vários vetores de movimento são determinados (o sinal *me_valid_out* é ativado por um período de relógio a cada novo vetor de movimento determinado).

O número de ciclos de relógio entre dois vetores de movimento varia dependendo dos blocos respectivos analisados, o que é um comportamento esperado para módulos de predição de movimento.

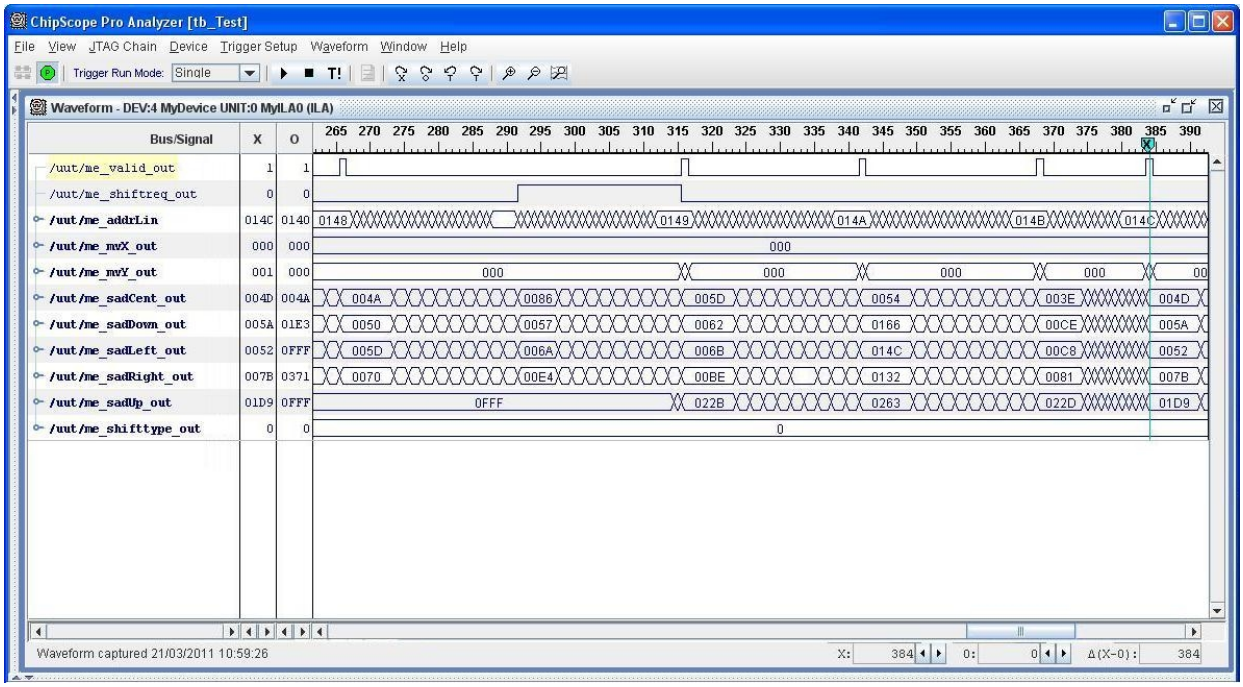


Figura 124: Tela capturada do Chipscope mostrando o módulo de predição em operação

Uma representação mais detalhada destes tempos de cálculo pode ser observado na Figura 125, que traz um trecho em destaque destacado do conjunto de sinais capturados no experimento anterior.

Percebe-se na figura a sinalização de dois vetores de movimento calculados pela arquitetura de hardware (um no instante 316 e outro em 342).

Assim, pode-se observar, na prática que a arquitetura proposta realizou o cálculo de um novo vetor de movimento em apenas 26 ciclos de relógio, o que comprova o alto desempenho da solução proposta.

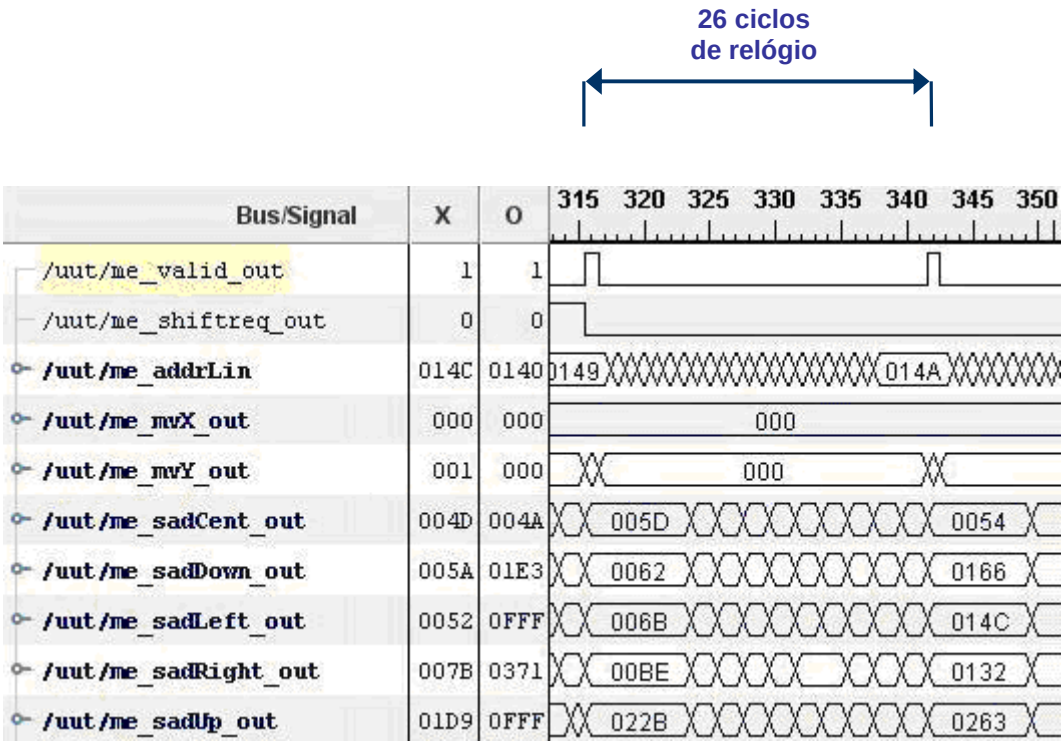


Figura 125: Visualização do tempo de cálculo de um vetor de movimento pela solução

7 CONCLUSÕES

O codificador de vídeo H.264/SVC é um padrão que une grande eficiência de codificação com aprimorada flexibilidade para operação em redes heterogêneas e, por isso, é aplicável a diversas situações (teleconferência, IPTV, entre outras). Sua implementação prática, entretanto, é muito difícil à elevada complexidade de seus algoritmos internos.

Considerando este cenário, o trabalho em questão, introduziu uma solução inovadora para a implementação prática de um codificador de vídeo escalável padrão H.264/SVC, baseado em um projeto cooperativo de hardware e software, também chamado de co-projeto de hardware e software.

O projeto realizado passou por três etapas importantes: modelagem, particionamento, integração e validação de forma prática levando em conta requisitos da aplicação e restrições da plataforma de hardware (interface de comunicação, freqüência de operação e barramentos de memória).

Para sua implementação prática, foram realizados diversos experimentos em laboratório sobre o software de referência H.264/SVC, a fim de facilitar o entendimento das reais demandas deste codificador. Os experimentos realizados permitiram a definição de um modelo refinado de codificador H.264/SVC, que incrementa significativamente seu desempenho, mantendo-se os mesmos padrões de qualidade, modelo este, que serviu de referência para este projeto, podendo contribuir também para outros trabalhos da área.

A partir deste modelo refinado, os diferentes algoritmos internos utilizados foram analisados tecnicamente, considerando-se questões como dependência de dados e potencial de paralelismo. Com base nesta análise, foi gerado um particionamento de módulos, onde somente os algoritmos mais propensos a paralelização (transformadas, quantização, filtragem e predição) foram portados para hardware. Demais funcionalidades foram mantidas como componentes em software.

A solução em hardware desenvolvida adota uma abordagem de reuso de módulos, que foi importante para reduzir a complexidade global da solução. Diferentes alternativas de projeto foram exploradas visando-se acoplamento com barramentos de 32 ou 64 bits, prevendo-se assim diferentes níveis de paralelismo e replicação de módulos.

A solução final apresentada, que inclui módulos otimizados para barramentos de 64 bits, apresentou desempenho compatível para suportar especificações de alta definição de um de um codificador escalável SNR (no todo, até 16 camadas de formato HD).

Todas as soluções desenvolvidas em hardware foram comparadas com diferentes trabalhos relacionados da literatura científica, constatando-se que as soluções desenvolvidas se mostravam como as mais eficientes, dentre todas as analisadas, principalmente considerando-se as limitações de barramentos de memória e interface de comunicação da plataforma de trabalho escolhida.

O codificador intra-quadro desenvolvido foi capaz de suportar o processamento de mais de 1.7Gamostras/s, o que ultrapassa o limite teórico de maior demanda escalável considerado para a aplicação alvo, ao mesmo tempo em que ocupa uma área de chip significativamente menor que outras soluções.

Para a validação da solução completa, foram realizados experimentos práticos, integrando-se a solução em hardware desenvolvida com uma aplicação em software baseada no código de referência JSVM, de forma que ambas as entidades (hardware e software) operassem de forma colaborativa, para permitir, em tempo de execução, a codificação escalável de diversas sequências de vídeo, caracterizando-se assim como uma inovadora solução prática de co-projeto de hardware e software para codificação H.264/SVC.

Para tanto, foi escolhida como plataforma de hardware uma placa de desenvolvimento com interface PCI Express, canal de comunicação utilizado para comunicação entre as diferentes entidades. Para aumento da eficiência da solução integrada foi utilizado um modelo de integração, onde pacotes de dados com vários macroblocos eram enviados por vez para a placa de hardware em um modelo de execução, que permitia que ambas as entidades pudessem trabalhar ao mesmo tempo.

Os resultados práticos obtidos após a integração da solução comprovam elevados ganhos de desempenho, quando comparados com uma solução unicamente por software.

Mais particularmente, o ganho registrado de desempenho para o protótipo desenvolvido para codificação intra foi de até 970% (9,7x vezes mais rápido), enquanto que a solução inter apresentou ganho de 675% (6,75x vezes mais rápido).

O módulo de predição desenvolvido, a partir de uma estratégia de busca zonal, que se utiliza de preditores de blocos vizinhos, se mostrou muito eficiente para a implementação de um codificador de vídeo escalável, uma vez que o primeiro ponto de partida já se encontra em média bem próximo do ponto final desejado. Esta escolha de projeto, em especial, se mostrou bem ajustada para atuação com um codificador escalável, uma vez que auxilia na busca pelo reaproveitamento de informações de outras camadas, inclusive de predição de movimento.

Além disso, a adoção das técnicas de sub-amostragem e truncamento de bit em conjunto com o padrão de diamante pequeno ajudaram a reduzir de forma significativa o tempo de acesso à memória (no melhor caso se consegue montar a matriz de registradores de um bloco de busca em topologia de diamante em apenas dez ciclos de relógio).

Ganhos de desempenho obtidos com este mecanismo de procura em diamante em relação a outros métodos mostram-se bem significativos (duas ou três vezes mais rápidos).

Em relação ao software de referência os ganhos obtidos da implementação em hardware são maiores que 1680%, ou seja maiores que 16,8 vezes para os experimentos feitos.

Apesar dos elevados ganhos obtidos, é importante destacar que ganhos ainda maiores podem ser atingidos se a plataforma de hardware for substituída por uma placa com interface de comunicação mais eficiente (por exemplo interface PCI Express 8x). Testes em laboratório comprovaram que a latência da placa escolhida (padrão PCI Express 1x) causa grande influência, restringindo a taxa efetiva de dados trocados entre as entidade de hardware e software.

Como trabalhos futuros se destaca a implementação e validação de novos módulos em hardware como o algoritmo compensador de movimento, assim como o módulo de refinamento para meio-pixel e quarto de pixel, os quais também devem trazer ganhos de desempenho para o sistema.

Também se propõe como trabalho futuro a realização de experimentos utilizando a mesma abordagem, porém migrando os módulos para outras plataformas, como arquiteturas do tipo NoC ou mesmo placas comerciais que possuam recursos de GPGPU, que se apresentam como soluções tecnologicamente promissoras.

REFERÊNCIAS

AGILENT, T. **PCI express performance measurements:** application note 1565. Santa Clara: Agilent, 2006. 8 p.

AGOSTINI, L. et al. High throughput architecture for H.264/AVC forward transforms block. In: ACM GREAT LAKES SYMPOSIUM ON VLSI, 16., 2006, Philadelphia. **Proceedings...** Philadelphia: ACM, 2006. p. 320-323.

_____. Design and FPGA prototyping of a H.264/AVC main profile decoder for HDTV. **Journal of SBC**, Porto Alegre, v. 12, p. 25-36, Sept. 2007.

AKPAN, V. A. **An introductory note on FPGA embedded system design technologies:** with an overview of Xilinx® Systems Design Tools. Thessaloniki: University of Thessaloniki, 2009. 31 p.

AMER, I.; BADAWY, W.; JULLIEN, G. A. High-performance hardware implementation of the H.264 simplified 8x8 transformation and quantization. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 30., 2005, Pensilvania. **Proceedings…** Pensilvania: ICASSP, 2005. p. 1137-1140.

ATITALLAK, A. B. et al. An FPGA implementation of HW/SW codesign architecture for H.263 video coding. In: RADHAKRISHNAN, S. **Effective Video Coding for Multimedia Applications**. Rijeka: Intech, 2011. p. 229-254.

BAE, T. M.; THANG, T. C.; RO, Y. M. Improvement of inter-layer motion prediction in scalable video. **IEICE Transactions on Information and System**, Tokyo, v. E90-D, n. 10, p. 1712-1715, Oct. 2007.

BERTOZZI, D. et al. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. **IEEE Transactions on Parallel and Distributed Systems**, Riverside, v. 16, n. 2, p. 113-129, Feb. 2005.

BILGIC, B. et al. Efficient integral image computation on the GPU. In: IEEE INTELLIGENT VEHICLES SYMPOSIUM, 4., 2010, San Diego. **Proceedings...** San Diego: ITSS, 2010. p. 528-533.

BOGEN, N. **4Q10 Set Top Box Database Update**. Scottsdale: In-stat, 2010. 34 p.

BROTHERTON, M. D.; BAYARD, D.; HANDS, D. S., Subjective quality assessment of the H.264/AVC in-loop de-blocking filter. **IEICE Transactions in Communication**, Tokyo, v. E89-B, n. 2, p. 273-280, Feb. 2006.

BRUNIG, M.; NIEHSEN, W. Fast full-search block matching. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v. 1, n. 2, p. 241–247, Feb. 2001.

BRUNO, G. **VEBIT**: um novo algoritmo para codificação de vídeo com escalabilidade. 2003. 95 f. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

CHEN, C-M.; CHEN, C-H. An efficient pipeline architecture for deblocking filter in H.264/AVC. **IEICE Transactions on Informatics and Systems**, Tokyo, v. E90-D, n. 1, p. 99-107, Jan. 2007.

CHEN, Y. K.; PENG, W. H. Implementation of real-time MPEG-4 FGS encoder. In: IEEE PACIFIC-RIM CONFERENCE ON MULTIMEDIA, 3., 2002, Hsinchu. **Proceedings...** Hsinchu: PCM, 2002. p. 839-846.

CHEN, S-J. et al. **Hardware software co-design of a multimedia SOC platform**. Taiwan: Springer, 2009. 172 p.

CHIEN, C-D. et al. A low-power motion compensation IP core design for MPEG-1/2/4 video decoding. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 2005, Kobe. **Proceedings...** Kobe: IEEE Circuits and Systems Society, 2005. p. 4542-4545.

COCK, J. D.; NOTEBAERT, S.; WALLE R. V. Transcoding from H.264/AVC to SVC with CGS Layers. In: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 14., 2007, San Antonio. **Proceedings...** San Antonio: IEEE Signal Processing Society, 2007. p. 73-76.

DARONCO, L. C. **Avaliação subjetiva de qualidade aplicada à codificação de vídeo escalável**. 2008. 101 f. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2008.

DAWEI, Y.; HOU, B.; CHUNHUI, Z. A fast mode decision algorithm of adaptive interlayer prediction in scalable video coding. In: INTERNATIONAL CONFERENCE ON INFORMATION, COMMUNICATIONS AND SIGNAL PROCESSING, 7., 2009, Macau. **Proceedings...** Macau: ICICS, 2009. p. 978-981.

DE MICHELI, G.; EMST, R.; WOLF, W. **Readings in hardware/software co-design.** San Francisco: Academic Press, 2002. 697 p.

DORNELLES, R. et al. Arquitetura de um módulo T dedicado à predição intra do padrão de compressão de vídeo H.264/AVC para uso no sistema brasileiro de televisão digital. Revistas eletrônicas – PUC-RS, Porto Alegre, v. 32, n. 62, p. 75-82, Jun. 2008.

_____. Transforms and quantization design targeting the H.264/AVC intra prediction constraints. In: Symposium on Integrated Circuits and Systems Design, 22., 2009, Natal Proceedings... Natal: SBCCI, 2009.

GAO, X. Q.; DUANMU, C. J.; ZOU, C. R. A multilevel successive elimination algorithm for block matching motion estimation. IEEE Transactions on Image Processing, Evanston, v. 9, p. 501–504, Mar. 2000.

GHANDI, M. M.; GHANBARI, M. The H.264/AVC video coding standard for the next generation multimedia communication. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v. 12, n. 7, p. 3-17, July 2003.

HE, W-F. VLSI implementation of full pixel motion estimation processor for MPEG-4 AS profile. In: IEEE CONFERENCE ON SOLID-STATE AND INTEGRATED CIRCUITS TECHNOLOGY, 7., 2004, Beijing. **Proceedings...** Beijing: Solid-State Circuits Society, 2004. p. 1633-1636.

HILLESTAD, O. I. et al. Adaptive H.264/MPEG-4 SVC video over IEEE 802.16 broadband wireless networks. In: INTERNATIONAL PACKET VIDEO WORKSHOP, 16., Lausanne. **Proceedings...** Lausanne: PV, 2007. p. 26-35.

HUANG, H-S.; PENG, W-H.; CHIANG, T. Advances in the scalable amendment of H.264/AVC. **Advances in Visual Content Analysis and Adaptation for Multimedia Communication**, New York, v. 45, n. 1, p. 68-76, Jan. 2007.

HUSEMANN, et al. **Recomendações técnicas da equipe RFP-03 para o SBTVD**: codificador e decodificador escalável MPEG-2. São Leopoldo: Unisinos, 2006a.

_____. Análise da implementação de algoritmos de codificação e decodificação de vídeo MPEG-2 HD escalável em hardware. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 33., 2006, Campo Grande. **Anais...** Campo Grande: SBC, 2006b, p. 16-31.

JERRAYA, A. A.; WOLF, W. Hardware/software interface codesign for embedded systems. **IEEE Computer**, Washington, v. 38, n. 2, p. 63-69, Feb. 2005.

JOINT SCALABLE VIDEO MODEL **H.264 SVC reference software (JSVM 9.19) and manual**. Berlim: Fraunhofer, 2010. 76 p.

JOINT VIDEO TEAM **Main Results of the AVC Complexity Analysis**: report W4964. Klagenfurt: JVT, 2002. 3 p. Disponível em: http:\www.asicfpga.com\site_upgrade\asicfpga\pds\image_pds_files\W4964.doc>. Acesso em: 20 mar. 2009.

KANT, S.; MITHUN, U.; GUPTA, P. Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications. In: INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS, 1., 2006, Las Vegas. **Proceedings...** Las Vegas: IEEE Consumer Electronics, 2006. p. 93-94.

KEITH, J. **Video desmystified.** 4. ed. Virginia: LLH Technology Publishing, 2004. 927 p.

KIM, M.; HWANG, I.; CHAE, S-I. A fast VLSI architecture for full-search variable block size motion estimation in MPEG-4 AVC/H.264. In: CONFERENCE ON ASIA SOUTH PACIFIC DESIGN AUTOMATION, 10., 2005, Shangai. **Proceedings...** Shangai: ASP-DAC, 2005. p. 631-634.

KIM, S. et al. Interframe coding using two-stage variable block-size multiresolution motion estimation and Wavelet decomposition. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v. 8, n. 4, p. 399-410, Aug. 1998.

KOGA, T. et al. Motion-compensated interframe coding for video conferencing. In: NATIONAL TELECOMMUNICATION CONFERENCE, 1981, New Orleans. **Proceedings...** New Orleans: NTC, 1981. p. G5.3.1-G5.3.5.

KORAH, R. et al. FPGA Implementation of integer transform and quantizer for H.264 encoder. **Journal of Signal Processing Systems,** New York, v. 53, n. 3, p. 261-269, Dec. 2008.

KUO, T-Y.; CHAN, C-H. Fast variable block size motion estimation for H.264 using likehood and correlation of motion field. **IEEE Transactions on Circuits and Systems for Video Technology,** New York, v. 16, n. 10, p. 1185-1195, Oct. 2006.

LAHTI, J. et al. Algorithmic optimization of H.264/AVC encoder. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 12., Island of Kos. Proceedings... Island of Kos: IEEE Circuits and System Society, 2005. p. 3463-3466.

LAI, Y-K.; CHEN, L-F.; CHIOU, W-C. A memory interleaving and interlacing architecture for deblocking filter. IEEE Transactions on Consumer Electronics. New York, v. 56, n. 4, p. 2812-2818, May 2010.

LEE, S. H. et al. Implementation of H.264/AVC decoder for mobile video applications. In: Conference Asia South Pacific Design Automation, 11., 2006, Yokohama. Proceedings...Yokohama: ASP-DAC, 2006. p. 120-121.

LI, B.; LIU, J. Multirate video multicast over the Internet: an overview. IEEE Network, New York, v. 17, n. 1, p. 24–29, Jan. 2003.

LI, L.; GOTO, S.; IKENAGA, T. A highly parallel architecture for deblocking filter in H.264/AVC. **IEICE Transactions on Information and Systems**, Tokyo, v. E88-D, n. 7, p. 1623-1629, July 2005.

LI, R.; ZENG, B.; LIOU, M. L. A new three-step search algorithm for block motion estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v. 4, n. 4, p. 438-442, Aug. 1994.

LI, Y.; HE, Y.; MEI. S. A highly parallel joint VLSI architecture for transforms in H.264/ AVC. **Journal of Signal Processing Systems**, New York, v. 50, n. 1, p. 19-32, Jan. 2008.

LIN, Y-L. S. **Essential issues in SOC design**: designing complex system-on-chip. Dordrecht: Springer, 2006. 411 p.

LIU, T-M. et al., In/post-loop deblocking filter with hybrid filtering schedule. **IEEE Transactions on Circuit and Systems for Video Technology**, New York, v. 17, n. 7, p. 937-943, July 2007.

LOGASHANMUGAN, E. An efficient hardware architecture for H.264 transform and quantization algorithms. **International Journal of Computer Science and Network Security**, Seul, v. 8, n. 6, p. 167-173, June 2008.

MARPE, D.; WIEGAND, T.; HERTZ, H. The H.264/MPEG4 advanced video coding standard and its applications. **IEEE Communications Magazine**, New York, v. 44, n. 8, p. 134-144, Aug. 2006.

MIN, K., CHONG, J. A memory and performance optimized architecture of deblocking filter in H.264/AVC. In: INTERNATIONAL CONFERENCE ON MULTIMEDIA AND

UBIQUITOUS ENGINEERING, 3., 2007, Seoul. **Proceedings...** Seoul: MUE, 2007. p. 220-225.

MPEG, F. **MPEG-4 Part 10/H.264 scalable video coding**: reports of IT Forum. Seoul: IT MPEG Forum, 2008. 59 p.

MURACH, M. et al. Optimal reconfigurable HW/SW co-design of load flow and optimal power flow computation. In: IEEE POWER ENGINEERING SOCIETY GENERAL MEETING, 2006, Quebec. **Proceedings...** Quebec: IEEE Power and Energy Society, 2006. p. 1-5.

MURALI, S. et al. Mapping and configuration methods for multi-use-case networks on chips. In: CONFERENCE ASIA SOUTH PACIFIC DESIGN AUTOMATION, 11., 2006, Yokohama. **Proceedings...** Yokohama: ASP-DAC, 2006. p. 146-151.

NAM, K. M. et al. A fast hierarchical motion vector estimation algorithm using mean pyramid. **IEEE Transactions on Circuit and Systems for Video Technology**, Tokyo, v. 5, n. 8, p. 344-351, Aug. 1999.

NARVEKAR, et al. An H.264/SVC memory architecture supporting spatial and course-grained quality scalabilities. In: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 16., 2009, Cairo. **Proceedings...** Cairo: IEEE Signal Processing Society, 2009. p. 2661-2664.

NDILI, O.; OGUNFUNMI, T. Efficient sub-pixel interpolation and low power VLSI architecture for fractional motion estimation in H.264/AVC. In: INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING AND COMMUNICATION SYSTEMS, 4., 2010, Gold Coast. Proceedings... Gold Coast: ICSPCS, 2010. p. 1-10.

NGUYEN, V-A.; TAN, Y-P. Efficient video transcoding from H.263 to H.264/AVC standard with enhanced rate control. EURASIP Journal on Applied Signal Processing, New York, v. 2006, p. 1-15, Jan. 2006.

NOTEBAERT, S.; COCK, J. D. **Hardware/software co-design of the H.264/AVC standard**: white paper. Gent: Ghent University, 2004. 2 p.

NUNO, R.; DIAS, T.; SOUZA, L. Customizable and reduced hardware motion estimation processors. In: LYSAGHT, P.; ROSENSTIEL, W. **New Algorithms, Architectures, and Applications for Reconfigurable Computing.** Dordrecht: Springer, 2005. p. 55-68.

OHM, J-R. Advances in scalable video coding. **Proceedings of the IEEE,** New York, v. 93, n. 1, p. 42-56, Jan. 2005.

OSORIO, R. R.; BRUGUERA, J. D. Arithmetic coding architecture for H.264/AVC CABAC compression system. In: EUROMICRO SYMPOSIUM ON DIGITAL SYSTEM DESIGN, 30., 2004, Rennes. **Proceedings...** Rennes: DSD, 2004. p. 62-69.

PARK, G-H.; YOO, W-H.; SUH, D-Y. H.264-based selective fine granular scalable video coding. **IEICE Transactions in Communication**, Tokyo, v. E89-B, n. 8, p. 2271-2274, Aug. 2006.

PASTUSZAK, G. Parallel symbol architectures for H.264/AVC binary coder based on arithmetic coding. In: SYMPOSIUM ON PARALLEL COMPUTING IN ELECTRICAL ENGINEERING, 5., 2006, Bialystok. **Proceedings...** Bialystok: PARELEC, 2006. p. 380-385.

PO, L. M.; MA, W. C. A novel four-step search algorithm for fast block motion estimation. **IEEE Transactions on Circuit and Systems for Video Technology**, New York, v. 6, n. 3, p. 313-317, June 1996.

PODILCHUCK, C. I.; JAYANT, N. S.; FARVARDIN, N. Three-dimensional subband coding of video. **IEEE Transactions on Image Processing**, Evanston, v. 4, n. 2, p. 125-139, Feb. 1995.

PORTO, M. **Arquiteturas de alto desempenho e baixo custo em hardware para a Estimação de movimento em vídeos digitais**. 2008. 101 f. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2008.

PRASOON, A. K.; RAJAN, K. 4x4 2-D DCT for H.264/AVC. In: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATION AND CONTROL, 2009, Mumbai. **Proceedings...** Mumbai: ICAC3, 2009. p. 573-577.

RADHA, H. M.; SCHAAR, M. V. D.; CHEN, Y. The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP. **IEEE Transactions on Multimedia**, New Jersey, v. 3, n. 1, p. 53-67, Mar. 2001.

RICHARDSON, I. E. G. **H.264 and MPEG-4 video compression**. Chichester: Wiley & Sons, 2003. 281 p.

RIECKL, J. **Scalable video for peer-to-peer streaming**. 2008. 53 p. Master thesis (Communication Engineering Master) – Institute of Communications and Radio-Frequency Engineering, University of Wien, Wien, 2008.

ROSA, V. S.; SUSIN, A. A.; BAMPI, S. A high performance H.264 deblocking filter. In: PACIFIC-RIM SYMPOSIUM ON IMAGE AND VIDEO TECHNOLOGY, 3., 2009, Gwangju. **Proceedings...** Gwangju: PSIVT, 2009. p. 955-964.

RUHIO-LOYOLA, J.; SALA, D.; ALI, A. I. Maximizing packet loss monitoring accuracy for reliable trace collections. In: IEEE WORKSHOP LOCAL AND METROPOLITAN AREA NETWORKS, 2008, Cluj-Napoca. **Proceedings...** Cluj-Napoca: LANMAN, 2008. p. 61-66.

SAYED, M.; BADAWI, W.; JULLIE, G. Towards an H.264/AVC HW/SW integrated solution: an efficient VBSME architecture. **IEEE Transactions on Circuits and Systems II**, New York, v. 55, n. 9, p. 912-916, Sept. 2008.

SCHAFER, R. et al. MCTF and scalability extension of H.264/AVC and its application to video transmission, storage and surveillance. In: IEEE VISUAL COMMUNICATIONS AND IMAGE PROCESSING, 2005, Beijing. **Proceedings...** Beijing: IEEE Circuits and Systems Society, 2005. 11 p.

SCHIERL, T. et al. Using H.264/AVC-based scalable video coding (SVC) for real time streaming in wireless IP networks. In: IEEE INTERNATIONAL SYMPOSIUM ON

CIRCUITS AND SYSTEMS, 2007, New Orleans. **Proceedings...** New Orleans: IEEE Circuits and Systems Society, 2007. p. 3455-3458.

SCHWARZ, H.; MARPE, D.; WIEGAND, T. Analysis of hierarchical B pictures and MCTF. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA & EXPO, 2006, Toronto. **Proceedings...** Toronto: ICME, 2006. p. 1929-1932.

SEGALL, C. A.; SULLIVAN, G. Spatial scalability within the H.264/AVC scalable video coding extension. **IEEE Transactions on Circuit and Systems for Video Technology**, Tokyo, v. 17, n. 9, p. 1121-1135, Sept. 2007.

SHIRANI, S.; KORDASIEWICZ R. C. ASIC and FPGA implementations of H.264 DCT and quantization blocks. In: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 12., 2005, Genoa. **Proceedings...** Genoa: IEEE Signal Processing Society, 2005. p. 1020-1023.

_____. On hardware implementation of DCT and quantization blocks for H.264/AVC. **Journal of VLSI Signal Processing**, New York, v. 47, n. 3, p. 189-199, June 2007.

SHIH, S-Y.; CHENG-RU, C.; YOUN-LONG, L. A near optimal deblocking filter for H.264 advanced video coding. In: ASIA SOUTH PACIFIC DESIGN AUTOMATION, 11., 2006, Yokohama. **Proceedings...** Yokohama: ASP-DAC, 2006. p. 170-175.

SILVA, A. M. C. et al. Exploração no espaço de projeto da Hadamard 4x4 direta do padrão de compressão de vídeo H.264/AVC. In: WORKSHOP IBERCHIP, 12., 2006, San Jose. **Proceedings...** San Jose: CYTED, 2006. p. 99-102.

SULLIVAN, G. J.; WIEGAND, T. Video compression – from concepts to the H.264/AVC. **Proceedings of the IEEE,** New York, v. 99, n. 1, p. 18-31, Jan. 2005.

TASDIZEN, O.; HAMZAOGLU, I. A high performance and low cost hardware architecture for H.264 transform and quantization algorithms. In: EUROPEAN SIGNAL PROCESSING CONFERENCE, 13., 2005, Antalya. **Proceedings...** Antalya: EUROSIPCO, 2005. p. 4-8.

TOURAPIS, A. M.; AU, O. C.; LIOU, M. L. Predictive motion vector field adaptive search technique (PMVFAST) – enhancing block based motion estimation. In: IEEE VISUAL COMMUNICATIONS AND IMAGE PROCESSING, 2001, San Jose. **Proceedings...** San Jose: IEEE Circuits and Systems Society, 2001. p. 883-892.

TOURAPIS, H. Y. C.; TOURAPIS, A. M. Fast motion estimation within the H.264 codec. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA & EXPO, 2003, Baltimore. **Proceedings...** Baltimore: ICME, 2003. p. 517-520.

WANG, K-Y. et al. 3-Stage pipelined deblocking filter. World Academy of Science, Engineering and Technology. New Mexico, v. 62, n. 38, p. 39-42, Feb. 2010.

WANG, T. et al. Parallel 4x4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 2003, Bangkok. **Proceedings...** Bangkok: IEEE Circuits and System Society, 2003. p. II800-II803.

WIEN, M.; SCHWARZ, H.; OELBAUM, T. Performance analysis of SVC. **IEEE Transactions on Circuit and Systems for Video Technology**, Tokyo, v. 17, n. 9, p. 1194-1203, Sept. 2007.

WOLF, W. A decade of HW/SW codesign. **IEEE Computer**, Washington, v. 36, n. 4, p. 38-43. Apr. 2003.

XILINX, I. **PCI 64/32 interface v3.0**: product specification. San Jose: Xilinx, 2005. 11 p.

_____. **LogiCORE IP endpoint block plus v1.14 for PCI express:** product specification. San Jose: Xilinx, 2010. 18 p.

XU, J.; CHEN, Z.; HE, Y. Efficient fast ME predictions and early-termination strategy based on H.264 statistical characters. In: INTERNATIONAL CONFERENCE ON INFORMATION, COMMUNICATION AND SIGNAL PROCESSING, 4., 2003, Singapore. **Proceedings...** Singapore: ICICS, 2003. p. 218-222.

YANG, S.; WOLF, W.; VIJAYKRISHNAN, N. Power and performance analysis of motion estimation based on hardware and software realizations. **IEEE Transactions on Computers**, Washington, v. 54, n. 6, p. 714-726, June 2005.

YANG, S-S. et al. A parallelism encoding framework for the temporal scalability of H.264/AVC scalable extension. In: IEEE INTERNATIONAL SYMPOSIUM ON MULTIMEDIA WORKSHOPS, 9., 2007, Taichung. **Proceedings...** Los Alamitos: IEEE Computer Society, 2007. p. 397-400.

YANG, W.; RATH, G.; GUILLEMOT, C. Improved interlayer prediction for scalable video coding. In: INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 32., 2007, Honolulu. **Proceedings...** Honolulu: ICASSP, 2007. p. 649-652.

YE, Q.; MCGREGOR, M. Hardware bottleneck evaluation and analysis of a software PC-based router. In: INTERNATIONAL SYMPOSIUM ON PERFORMANCE EVALUATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS, 2008, Edinburgh. **Proceedings...** Edinburgh: SPECTS, 2008. p. 480-487.

ZAN, J.; AHAMAD, M. O.; SWAMY, M. N. S. Pyramidal motion estimation techniques exploiting intra-level motion correlation. **IEEE Transactions on Circuits and Systems II**, New York, v. 50, n. 2, p. 83-93, Feb. 2003.

ZANDONAI, D.; BAMPI, S.; BERGERMAN, M. ME64 - a parallel hardware architecture for motion estimation implemented in FPGA. In: IFIP CONFERENCE ON PARALLEL AND DISTRIBUTED EMBEDDED SYSTEMS, 2., 2004, Toulouse. **Proceedings...** Toulouse: DIPES, 2004. p. 317-326.

ZHANG, Y-Q.; ZAFAR, S. Motion-compensated wavelet transform coding for color video compression. **IEEE Transactions on Circuit and Systems for Video Technology**, Tokyo, v. 2, n. 3, p. 285–296, Sept. 1992.

ZHAO, Z.; LIANG, P. A highly efficient parallel algorithm for H.264 video encoder. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, 31., 2006, Toulouse. **Proceedings...** Toulouse: ICASSP, 2006. p. V.489-V.492.

ZHU, S.; MA, K-K. A new diamond search algorithm for fast block-matching motion estimation. **IEEE Transactions on Image Processing**, Piscataway, v. 9, n. 2, p. 287-290, Feb. 2000.

APÊNDICE A

Resultados de ensaios realizados sobre software de referência H.264/SVC

Tabela 29 – Resultados comparativos entre codificações escaláveis SVC do tipo temporal

Seqüência	Cenário	Tempo de execução	Cálculo de PSNR sobre vídeo reconstruído		
			Y	U	V
BUS	GOP 8 (T0, T1, T2 e T3)	5m4,61s	26,9276	37,6701	38,8766
	GOP 16 (T0, T1, T2, T3 e T4)	4m56,59s	27,1269	37,8002	39,1702
CITY	GOP 8 (T0, T1, T2 e T3)	7m17,32s	28,2216	40,1815	41,3044
	GOP 16 (T0, T1, T2, T3 e T4)	8m4,73s	29,0424	40,4272	41,6763
FOOTBALL	GOP 8 (T0, T1, T2 e T3)	11m46,16s	27,9469	35,6685	38,0204
	GOP 16 (T0, T1, T2, T3 e T4)	12m35,53s	28,0186	35,8302	38,0530
FOREMAN	GOP 8 (T0, T1, T2 e T3)	9m14,99s	29,7561	37,7503	38,7307
	GOP 16 (T0, T1, T2, T3 e T4)	10m3,05s	30,2318	38,0815	39,1268
MOBILE	GOP 8 (T0, T1, T2 e T3)	6m51,95s	26,0473	31,9799	31,5073
	GOP 16 (T0, T1, T2, T3 e T4)	7m21.9s	26,7310	32,4281	31,8720

Tabela 30 – Resultados comparativos entre codificações escaláveis SVC do tipo espacial

Seqüência	Cenário	Tempo sem conversão	Tempo de conversão	Cálculo de PSNR sobre o vídeo final reconstruído		
				Y	U	V
BUS	ESS desativado	14m44s	-	28,4259	38,4942	40,0358
	ESS caso diádico	18m56s	1,78s	28,6376	38,5018	39,9537
	ESS caso não diádico	20m10s	1,30s	28,6231	38,5242	39,8948
CITY	ESS desativado	22m12s	-	28,1680	40,2949	41,5393
	ESS caso diádico	27m33s	3,55s	30,0887	41,1585	42,5048
	ESS caso não diádico	28m25s	2,64s	30,0017	41,1251	42,4730
FOOTBALL	ESS desativado	33m23s	-	29,1408	36,6907	38,7927
	ESS caso diádico	44m10s	3,14s	29,3352	36,6966	38,6397
	ESS caso não diádico	52m41s	2,245s	29,2427	36,5349	38,7618
FOREMAN	ESS desativado	27m15s	-	31,1650	38,7790	39,9554
	ESS caso diádico	34m32s	3,63s	31,2439	38,7229	39,8473
	ESS caso não diádico	36m29s	2,61s	31,1932	38,6459	39,8469
MOBILE	ESS desativado	20m58s	-	27,8095	33,0881	32,6796
	ESS caso diádico	26m58s	3,65s	27,9262	33,0494	32,5799
	ESS caso não diádico	27m47s	2,59s	27,9125	33,0144	32,5956

Tabela 31 – Resultados comparativos entre codificações escaláveis SVC do tipo SNR

Seqüência	Cenário	Tempo de execução	Cálculo de PSNR sobre vídeo da camada 0			Cálculo de PSNR sobre vídeo da camada 1		
			Y	U	V	Y	U	V
BUS	MGS	19m26s	28,7193	38,5612	40,0321	28,7423	38,5838	40,0343
	CGS	19m4s	26,8880	37,7900	39,0486	28,4246	38,5216	40,0619
CITY	MGS	29m39s	30,1623	41,0502	42,3906	30,1918	41,0852	42,3963
	CGS	29m576s	28,1680	40,2949	41,5393	30,0317	41,1263	42,4451
FOOTBALL	MGS	43m5s	29,5533	36,7665	38,8445	29,5832	36,7989	38,8907
	CGS	45m423s	27,9219	35,7175	37,8998	29,1680	36,7123	38,7593
FOREMAN	MGS	35m12s	31,4965	38,6786	39,8924	31,5466	38,7060	39,9352
	CGS	36m382s	29,6952	37,8945	38,8463	31,1929	38,7921	39,9472
MOBILE	MGS	28m12s	27,9978	33,0384	32,6003	28,0168	33,1168	32,6898
	CGS	28m197s	26,0387	31,9959	31,5175	27,8107	33,0938	32,6790

Tabela 32 – Resultados comparativos entre configurações do módulo de predição

Seqüência	Cenário	Tempo de execução	Cálculo de PSNR sobre vídeo da camada 0			Cálculo de PSNR sobre vídeo da camada 1		
			Y	U	V	Y	U	V
BUS	Normal	12m29s	31,2334	39,4368	41,1096	31,2242	39,4413	41,1065
	Busca completa	506m28s	31,2599	39,5160	41,2126	31,2511	39,5190	41,2123
	Sem bipredição	9m29s	31,1962	39,4349	41,1104	31,1870	39,4388	41,1072
	Janela busca reduzida	7m23s	31,2080	39,4250	41,0440	31,2001	39,4258	41,0447
	Janela busca expandida	21m17s	31,2172	39,4404	41,1422	31,2076	39,4431	41,1422
CITY	Normal	19m18s	32,2839	41,9664	43,3604	32,2301	41,9632	43,3577
	Busca completa	1072m41s	32,2854	41,9927	43,3959	32,2233	41,9935	43,3948
	Sem bipredição	12m39s	32,2772	41,9663	43,3583	32,2246	41,9632	43,3555
	Janela busca reduzida	14m25s	32,2722	41,9591	43,3417	32,2224	41,9580	43,3386
	Janela busca expandida	26m8s	32,2816	41,9434	43,3799	32,2337	41,9418	43,3816
FOOTBALL	Normal	28m19s	32,0230	38,2511	40,1090	32,0217	38,2551	40,1109
	Busca completa	837m2s	32,0429	38,2843	40,1195	32,0401	38,2877	40,1220
	Sem bipredição	23m55s	31,9995	38,2483	40,0959	31,9984	38,2518	40,0977
	Janela busca reduzida	12m49s	31,9882	38,2345	40,0924	31,9893	38,2368	40,0931
	Janela busca expandida	55m5s	32,0184	38,2236	40,1395	32,0132	38,2279	40,1412
FOREMAN	Normal	22m42s	33,4828	39,7403	41,1286	33,4723	39,7394	41,1241
	Busca completa	1055m20s	33,5349	39,8036	41,1287	33,5187	39,7956	41,1294
	Sem bipredição	16m34s	33,4566	39,7463	41,1381	33,4468	39,7456	41,1346
	Janela busca reduzida	14m27s	33,4912	39,7281	41,1049	33,4768	39,7224	41,1064
	Janela busca expandida	36m13s	33,4788	39,7848	41,1574	33,4652	39,7838	41,1502

Tabela 33 – Resultados comparativos entre resoluções do módulo de predição

Seqüência	Cenário	Tempo de execução	Valor de PSNR					
			Camada 0			Camada 1		
			Y	U	V	Y	U	V
BUS	A	4m52s	31,1305	39,4044	41,1280	31,1225	39,4095	41,1301
	B	9m25s	31,1871	39,3909	41,1252	31,1795	39,3923	41,1274
	C	11m18s	31,1940	39,4408	41,1843	31,1834	39,4449	41,1830
	D	1m43s	30,9059	39,3111	41,0275	30,8994	39,3116	41,0303
	E	0m27s	29,9308	38,5876	40,0120	29,9389	38,5886	40,0060
	F	9m49s	30,3994	39,3681	40,9918	30,3920	39,3685	40,9919
	G	11m13s	31,0004	39,3710	41,0689	30,9945	39,3733	41,0676
	H	9m47s	30,3994	39,3681	40,9918	30,3920	39,3685	40,9919
	I	3m44s	30,3703	39,3217	40,9539	30,3599	39,3226	40,9540
	J	7m36s	30,3570	39,3970	40,9995	30,3529	39,3992	40,9985
	K	8m56s	30,3466	39,3783	41,0137	30,3415	39,3824	41,0147
	L	1m22s	30,2412	39,2366	40,9058	30,2357	39,2372	40,9055
	M	0m27s	29,9308	38,5876	40,0120	29,9389	38,5886	40,0060
	N	4m15s	30,9446	39,3992	41,0857	30,9381	39,3988	41,0839
	O	8m33s	30,9567	39,3807	41,0339	30,9534	39,3810	41,0343
	P	10m6s	30,9560	39,4024	41,0906	30,9500	39,4044	41,0921
	Q	1m31s	30,7545	39,2669	40,9699	30,7490	39,2693	40,9711
	R	0m27s	29,9308	38,5876	40,0120	29,9389	38,5886	40,0060
	S	3m45s	30,3703	39,3217	40,9539	30,3599	39,3226	40,9540
	T	7m31s	30,3570	39,3970	40,9995	30,3529	39,3992	40,9985
	U	8m58s	30,3466	39,3783	41,0137	30,3415	39,3824	41,0147
	V	1m22s	30,2412	39,2366	40,9058	30,2357	39,2372	40,9055
	W	0m27s	29,9308	38,5876	40,0120	29,9389	38,5886	40,0060
CITY	A	7m6s	32,2821	41,9656	43,3456	32,2243	41,9637	43,3482
	B	14m51s	32,2534	41,9632	43,3352	32,1942	41,9629	43,3322
	C	17m47s	32,1713	41,9470	43,3398	32,1239	41,9470	43,3394
	D	2m33s	32,1516	41,9384	43,3034	32,0806	41,9385	43,2974
	E	0m49s	30,1613	40,9967	42,4002	30,1621	40,9967	42,4002
	F	14m31s	31,5996	41,7018	43,1170	31,5680	41,7042	43,1142
	G	17m5s	32,1877	41,8570	43,3065	32,1011	41,8520	43,3007
	H	14m30s	31,5996	41,7018	43,1170	31,5680	41,7042	43,1142
	I	5m6s	31,5711	41,6571	43,1126	31,5344	41,6619	43,1128
	J	11m26s	31,5455	41,6980	43,1051	31,5187	41,6963	43,1039
	K	13m25s	31,4916	41,7032	43,1222	31,4605	41,7036	43,1229
	L	1m58s	31,4563	41,6592	43,0973	31,4277	41,6601	43,0978
	M	0m49s	30,1613	40,9967	42,4002	30,1621	40,9967	42,4002
	N	6m7s	32,1773	41,8509	43,2712	32,1179	41,8492	43,2717
	O	13m16s	32,1431	41,8660	43,3181	32,0895	41,8661	43,3153

	P	15m48s	32,0733	41,8668	43,3030	32,0309	41,8665	43,3036
	Q	2m16s	32,0377	41,8713	43,2599	31,9683	41,8722	43,2612
	R	0m49s	30,1613	40,9967	42,4002	30,1621	40,9967	42,4002
	S	5m9s	31,5711	41,6571	43,1126	31,5344	41,6619	43,1128
	T	11m22s	31,5455	41,6980	43,1051	31,5187	41,6963	43,1039
	U	13m20s	31,4916	41,7032	43,1222	31,4605	41,7036	43,1229
	V	1m58s	31,4563	41,6592	43,0973	31,4277	41,6601	43,0978
	W	0m50s	30,1613	40,9967	42,4002	30,1621	40,9967	42,4002
FOOTBALL	A	12m6s	31,9827	38,2038	40,0824	31,9790	38,2075	40,0842
	B	20m36s	32,0025	38,2670	40,1246	31,9906	38,2697	40,1148
	C	24m52s	31,9570	38,2461	40,1023	31,9573	38,2479	40,1032
	D	4m14s	31,9314	38,1326	40,0152	31,9283	38,1320	40,0124
	E	0m44s	31,8068	38,0547	39,8879	31,8216	38,0564	39,8889
	F	24m51s	31,7927	38,2196	40,0743	31,7956	38,2228	40,0748
	G	27m12s	31,9422	38,2821	40,1437	31,9311	38,2724	40,1395
	H	24m40s	31,7927	38,2196	40,0743	31,7956	38,2228	40,0748
	I	10m43s	31,7881	38,1309	40,0487	31,7901	38,1334	40,0505
	J	17m52s	31,7904	38,1905	40,0588	31,7927	38,1933	40,0592
	K	21m44s	31,7291	38,2184	40,0308	31,7373	38,2201	40,0357
	L	3m48s	31,7820	38,1253	40,0442	31,7845	38,1263	40,0463
	M	0m43s	31,8068	38,0547	39,8879	31,8216	38,0564	39,8889
	N	11m34s	31,9277	38,2039	40,0828	31,9269	38,2062	40,0814
	O	19m36s	31,9341	38,2326	40,1287	31,9331	38,2386	40,1309
	P	23m47s	31,8943	38,2304	40,0665	31,8958	38,2288	40,0619
	Q	4m03s	31,8848	38,1384	40,0075	31,8767	38,1397	40,0081
	R	0m43s	31,8068	38,0547	39,8879	31,8216	38,0564	39,8889
	S	10m42s	31,7881	38,1309	40,0487	31,7901	38,1334	40,0505
	T	17m50s	31,7904	38,1905	40,0588	31,7927	38,1933	40,0592
	U	21m45s	31,7291	38,2184	40,0308	31,7373	38,2201	40,0357
	V	3m48s	31,7820	38,1253	40,0442	31,7845	38,1263	40,0463
	W	0m43s	31,8068	38,0547	39,8879	31,8216	38,0564	39,8889
FOREMAN	A	9m3s	33,4493	39,7462	41,0645	33,4376	39,7474	41,0669
	B	17m04s	33,4403	39,7412	41,1067	33,4285	39,7361	41,1035
	C	20m21s	33,4007	39,7452	41,1287	33,3989	39,7428	41,1322
	D	3m19s	33,2904	39,6405	40,9893	33,2765	39,6377	40,9856
	E	0m46s	31,8697	39,0170	40,2677	31,8908	39,0174	40,2713
	F	18m53s	33,0498	39,6486	40,9773	33,0349	39,6411	40,9701
	G	21m3s	33,3840	39,7139	41,1012	33,3705	39,7108	41,0993
	H	18m47s	33,0498	39,6486	40,9773	33,0349	39,6411	40,9701
	I	7m30s	33,0207	39,6059	40,9509	33,0161	39,6041	40,9452
	J	14m13s	32,9805	39,6037	40,9402	32,9837	39,6050	40,9371
	K	16m59s	32,9478	39,6328	40,9688	32,9521	39,6248	40,9644
	L	2m48s	32,8652	39,5192	40,8730	32,8634	39,5134	40,8629

	M	0m45s	31,8697	39,0170	40,2677	31,8908	39,0174	40,2713
	N	8m20s	33,3628	39,6938	41,0288	33,3265	39,6940	41,0262
	O	15m51s	33,3060	39,6806	41,0549	33,2916	39,6785	41,0528
	P	19m2s	33,2841	39,6767	41,0675	33,2817	39,6720	41,0584
	Q	3m06s	33,1869	39,6034	40,9474	33,1682	39,5960	40,9456
	R	0m46s	31,8697	39,0170	40,2677	31,8908	39,0174	40,2713
	S	7m31s	33,0207	39,6059	40,9509	33,0161	39,6041	40,9452
	T	14m11s	32,9805	39,6037	40,9402	32,9837	39,6050	40,9371
	U	16m56s	32,9478	39,6328	40,9688	32,9521	39,6248	40,9644
	V	2m48s	32,8652	39,5192	40,8730	32,8634	39,5134	40,8629
	W	0m46s	31,8697	39,0170	40,2677	31,8908	39,0174	40,2713
MOBILE	A	6m20s	30,1558	34,2477	33,9449	30,0613	34,2318	33,9379
	B	15m10s	30,1890	34,2993	33,9757	30,1192	34,2938	33,9663
	C	17m16s	30,1985	34,3063	34,0009	30,1397	34,2935	33,9896
	D	2m27s	29,9932	34,2085	33,8973	29,9068	34,1992	33,8892
	E	0m56s	28,1754	33,4147	33,0702	28,1763	33,4146	33,0704
	F	15m45s	29,3725	34,5759	34,2308	29,3618	34,5669	34,2228
	G	17m11s	30,0628	34,3335	34,0054	30,0095	34,3193	33,9996
	H	15m38s	29,3725	34,5759	34,2308	29,3618	34,5669	34,2228
	I	5m11s	29,3162	34,5548	34,2120	29,2985	34,5454	34,2024
	J	12m34s	29,3444	34,5809	34,2167	29,3372	34,5702	34,2124
	K	14m23s	29,3505	34,5915	34,2269	29,3374	34,5830	34,2226
	L	2m04s	29,2347	34,5360	34,1522	29,2244	34,5276	34,1460
	M	0m56s	28,1754	33,4147	33,0702	28,1763	33,4146	33,0704
	N	5m40s	29,9983	34,2969	33,9798	29,9307	34,2856	33,9618
	O	13m56s	30,0283	34,3082	33,9977	29,9880	34,2991	33,9894
	P	15m46s	30,0236	34,3284	34,0099	29,9708	34,3179	34,0045
	Q	2m14s	29,8708	34,2432	33,9440	29,8396	34,2316	33,9327
	R	0m56s	28,1754	33,4147	33,0702	28,1763	33,4146	33,0704
	S	5m11s	29,3162	34,5548	34,2120	29,2985	34,5454	34,2024
	T	12m33s	29,3444	34,5809	34,2167	29,3372	34,5702	34,2124
	U	14m27s	29,3505	34,5915	34,2269	29,3374	34,5830	34,2226
	V	2m04s	29,2347	34,5360	34,1522	29,2244	34,5276	34,1460
	W	0m56s	28,1754	33,4147	33,0702	28,1763	33,4146	33,0704

Legenda

Cenário					
A	Disable 8x8				
B	Disable 16x8				
C	Disable 16x16				
D	Disable 8x8	Disable 16x8			
E	Disable 8x8	Disable 16x8	Disable 16x16		
F	Disable HPEL				

G	Disable QPEL				
H	Disable HPEL	Disable QPEL			
I	Disable HPEL	Disable 8x8			
J	Disable HPEL	Disable 16x8			
K	Disable HPEL	Disable 16x16			
L	Disable HPEL	Disable 8x8	Disable 16x8		
M	Disable HPEL	Disable 8x8	Disable 16x8	Disable 16x16	
N	Disable QPEL	Disable 8x8			
O	Disable QPEL	Disable 16x8			
P	Disable QPEL	Disable 16x16			
Q	Disable QPEL	Disable 8x8	Disable 16x8		
R	Disable QPEL	Disable 8x8	Disable 16x8	Disable 16x16	
S	Disable HPEL	Disable QPEL	Disable 8x8		
T	Disable HPEL	Disable QPEL	Disable 16x8		
U	Disable HPEL	Disable QPEL	Disable 16x16		
V	Disable HPEL	Disable QPEL	Disable 8x8	Disable 16x8	
W	Disable HPEL	Disable QPEL	Disable 8x8	Disable 16x8	Disable 16x16

Tabela 34 – Resultados comparativos de módulos de entropia no codificador escalável svc

Seqüência	Cenário	Tempo de execução	Cálculo de PSNR sobre vídeo da camada 0			Cálculo de PSNR sobre vídeo da camada 1		
			Y	U	V	Y	U	V
BUS	CAVLC	19m21s	28,7193	38,5612	40,0321	28,7423	38,5838	40,0343
	CABAC	19m59s	26,8880	37,7900	39,0486	28,4246	38,5216	40,0619
CITY	CAVLC	26m35s	29,5320	40,8312	42,0271	29,6080	40,8866	42,0892
	CABAC	30m01s	28,1680	40,2949	41,5393	30,0317	41,1263	42,4451
FOOTBALL	CAVLC	38m28s	29,5354	36,7800	38,6327	29,5804	36,8461	38,7525
	CABAC	44m34s	27,9219	35,7175	37,8998	29,1680	36,7123	38,7593
FOREMAN	CAVLC	31m52s	31,1008	38,3972	39,6127	31,1775	38,4169	39,6383
	CABAC	36m41s	29,6952	37,8945	38,8463	31,1929	38,7921	39,9472
MOBILE	CAVLC	25m48s	27,5238	32,5359	32,1845	27,5715	32,6947	32,3403
	CABAC	28m23s	26,0387	31,9959	31,5175	27,8107	33,0938	32,6790

Tabela 35 – Resultados comparativos do uso de filtro MCTF no codificador escalável svc

Seqüência	Cenário	Tempo de execução	Cálculo de PSNR sobre vídeo da camada 0			Cálculo de PSNR sobre vídeo da camada 1		
			Y	U	V	Y	U	V
BUS	normal	14m44s	26,8880	37,7900	39,0486	28,4259	38,4942	40,0358
	MCTF	19m25s	31,2517	38,8499	39,9902	28,5109	38,6196	39,9821
CITY	normal	22m12s	28,1680	40,2949	41,5393	28,1680	40,2949	41,5393
	MCTF	29m08s	28,2739	40,2879	41,5168	30,1328	41,2046	42,4129
FOOTBALL	normal	33m23s	27,9219	35,7175	37,8998	29,1408	36,6907	38,7927
	MCTF	45m11s	27,9364	35,7657	37,9868	29,2000	36,7816	38,7411
FOREMAN	normal	27m15s	29,6952	37,8945	38,8463	31,1650	38,7790	39,9554
	MCTF	35m38s	29,7390	37,9474	38,8880	31,2517	38,8499	39,9902
MOBILE	normal	20m58s	26,0387	31,9959	31,5175	27,8095	33,0881	32,6796
	MCTF	28m45s	26,1438	32,4333	31,9793	26,1438	32,4333	31,9793