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

LUCIANO VOLCAN AGOSTINI

Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Dr. Sergio Bampi
Orientador

Prof. Dr. Ivan Saraiva da Silva (UFRN)
Co-orientador

Porto Alegre, agosto de 2007.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Agostini, Luciano Volcan

Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC / Luciano Volcan Agostini – Porto Alegre: Programa de Pós-Graduação em Computação, 2007.

172 f.:il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2007. Orientador: Sergio Bampi; Co-orientador: Ivan Saraiva da Silva.

1.Codificação de Vídeo. 2.Padrão H.264/AVC. 3.Arquiteturas VLSI. I. Bampi, Sergio. II. Silva, Ivan Saraiva da. III. Título.

Alcei a perna no pingo
E saí sem rumo certo
Olhei o pampa deserto
E o céu fincado no chão
Troquei as rédeas de mão
Mudei o pala de braço
E vi a lua no espaço
Clareando todo o rincão

E a trotezito no mais
Fui aumentando a distância
Deixar o rancho da infância
Coberto pela neblina
Nunca pensei que minha sina
Fosse andar longe do pago
E trago na boca o amargo
Dum doce beijo de china

Sempre gostei da morena
É a minha cor predileta
Da carreira em cancha reta
Dum truco numa carona
Dum churrasco de mamona
Na sombra do arvoredo
Onde se oculta o segredo
Num teclado de cordeona

Cruzo a última cancela
Do campo pro corredor
E sinto um perfume de flor
Que brotou na primavera.
À noite, linda que era,
Banhada pelo luar
Tive ganas de chorar
Ao ver meu rancho tapera

Como é linda a liberdade
Sobre o lombo do cavalo
E ouvir o canto do galo
Anunciando a madrugada
Dormir na beira da estrada
Num sono largo e sereno
E ver que o mundo é pequeno
E que a vida não vale nada
(...)

Falam muito no destino
Até nem sei se acredito
Eu fui criado solito
Mas sempre bem prevenido
Índio do queixo torcido
Que se amansou na experiência
Eu vou voltar pra querência
Lugar onde fui parido

João da Cunha Vargas

AGRADECIMENTOS

Gostaria de agradecer, em primeiro lugar, ao povo brasileiro, que financiou meus estudos durante toda a minha vida escolar e acadêmica. Este povo, que por vezes vê a esperança de uma vida melhor se diluir em tantas injustiças cotidianas, mas que ainda assim resiste e luta, merece um país melhor, mais justo, mais humano. Espero que, na minha vida profissional, eu consiga contribuir para que estes objetivos sejam atingidos.

Devo muito em minha vida à pessoa que escolhi para ser minha parceira e companheira nesta caminhada. Sem ela, nada disso seria possível. É uma alegria enorme saber que tenho alguém que posso contar nos momentos bons e ruins, alguém que está sempre ao meu lado para incentivar e motivar, alguém que, mesmo com as dificuldades do dia a dia, consegue ser a luz do meu caminho. Cris, sei que este muito obrigado não contempla tudo o que tenho para te agradecer, mas é a forma de deixar registrado neste papel o quanto sou grato por existires, por estares comigo, por seres a mãe de meus filhos. Eu te amo muito!

Meus filhos, Lucas, Isabela e Juliana, são, talvez, os que mais sofreram com minha ausência neste período em que eu estava investindo na minha formação. Afinal, noites, madrugadas, manhãs e tardes são, para um doutorando, turnos normais de trabalho. A meus filhos, que são a essência da minha razão de viver, prosperar e ser feliz, mais que um obrigado, eu gostaria de deixar um pedido de desculpas, por ter estado tão pouco presente neste último período. Espero poder compensar esta ausência com muito amor, dedicação, atenção e carinho nos próximos anos.

Também sou muito grato ao meu pai, Accelino, e à minha mãe, Neusa, que me geraram e me criaram com muito amor. Foram eles que me ensinaram a maior parte dos valores que trago comigo até hoje e, então, contribuíram diretamente para mais esta conquista. Minha mãe, sempre preocupada com a nossa saúde e sempre disposta a ajudar em momentos difíceis que passamos com as inevitáveis doenças das crianças, foi de extrema importância para suportarmos os percalços do caminho. Meu pai, sempre pronto para ajudar, seja em um corte de grama, seja em um churrasco, além de estar sempre disponível para uma conversa consistente sobre teologia, sobre política ou sobre futebol, sempre foi e é a referência de toda a família. Referência de valores e de princípios. É incrível como fico mais tranqüilo apenas em saber que ele está por perto.

Ao prof. Bampi e ao prof. Ivan, meus orientadores e amigos, tenho muito que agradecer. Fazem já mais de sete anos que convivemos e, deste convívio, tirei muitas lições. Confesso que sinto enorme admiração e carinho por estes dois mestres que tive e tenho em minha vida. Busco hoje, como professor, seguir os seus passos, na esperança de um dia contribuir com a formação de meus alunos da mesma maneira com que meus orientadores contribuíram com a minha formação, tanto técnica quanto humana. Talvez a maior contribuição de ambos para mim foi o esforço que fizeram para me demover da idéia de desistir do meu doutorado. Foram estes dois amigos que não permitiram que eu abandonasse o curso após ter sido severamente criticado quando da defesa de meu exame de qualificação. Aquele foi um momento triste, que me fez entrar em profunda depressão e a deixar de acreditar nas minhas capacidades. Mas a ajuda, incentivo e apoio do Bampi e do Ivan permitiram que eu me levantasse novamente e chegasse até o final deste desafio.

À Universidade Federal de Pelotas, instituição que me acolheu como professor e que, passados pouco mais de 24 meses de meu ingresso, me liberou para a conclusão de meu doutorado. Sem esta liberação, meu doutorado teria sido inviabilizado, então, tenho muito que agradecer à instituição e a seus administradores, que foram sensíveis quanto ao meu afastamento. A UFPel é, além de minha fonte de renda, minha fonte de satisfação profissional. Mesmo com todos os problemas e dificuldades, é lá que eu sinto que posso realmente dar uma contribuição efetiva para a sociedade. Lá eu percebo que minha atuação é muito importante para as conquistas coletivas, pois há ainda muito por fazer. É muito fácil trabalhar e produzir onde tudo está pronto. É fácil e chato. Mas produzir em um local com tantos desafios como a UFPel é atividade instigante e desafiadora.

Os órgãos de fomento à pesquisa do Brasil também foram muito importantes na conclusão de meu doutorado e a isto sou muito grato. O CNPq, além de financiar minha bolsa através do Programa Nacional de Microeletrônica, tem apoiado as pesquisas que realizo através de diversos projetos, entre os quais merecem ser destacados três projetos: o SoCMicro (cooperação entre UFRGS, UFPel e UCPel), financiado pelo programa PDI-TI; o projeto APISE (cooperação entre UFPel e FURG), financiado pelo programa PDPG-TI e o projeto SOC-Reuse, financiado pelo Edital Universal. A FINEP também tem contribuição importante com a pesquisa desenvolvida nesta tese a partir do financiamento do consórcio H.264/AVC Brasil (ligado às pesquisas relativas ao Sistema Brasileiro de Televisão Digital) que, na UFRGS, contou com mais de uma dezena de pesquisadores. Finalmente, à FAPERGS, que com os últimos recursos antes do bloqueio de financiamento por parte do governo do RS, aprovou um projeto de minha autoria no Edital PROADE3 que, mesmo com recursos mais escassos, também contribuiu com a execução deste trabalho.

Meus colegas da UFRGS, com os quais passei tantas horas nos últimos anos, também merecem meu agradecimento. Foi a partir da interação com estes amigos que diversas idéias exploradas na minha tese surgiram e foi compartilhando momentos de descontração e de trabalho que conseguimos todos atingir resultados tão interessantes, tanto em termos de publicação de artigos, quanto em termos de desenvolvimento tecnológico. Mas, talvez o melhor resultado desta integração tenha sido a construção dos nossos laços de amizade que, espero, não se desfaçam facilmente. Meu muito obrigado a todos, mas especialmente à Aninha, ao Vagner, ao Arnaldo, à Thaísa, ao Marcelo, ao Roger e ao Bruno.

Além de meu orientador, todos os demais professores do GME da UFRGS também foram muito importantes para minha formação e sou muito grato a todos. Gostaria de agradecer, especialmente, ao professores Susin e Reis, que foram e são exemplos de pesquisadores e parceiros em diversas aventuras intelectuais e humanas.

Sou, também, muito grato aos colegas da UFPel, que suportaram a minha ausência por quase três anos e, neste período, ficaram sobrecarregados de atividades em função de minha ausência. Sou grato a eles, pois só consegui concluir meu doutorado porque estava afastado das atividades na UFPel. Em especial, gostaria de agradecer ao professor Güntzel, que hoje está trabalhando na UFSC, por ter sido um parceiro de trabalho formidável em pesquisa e em ensino. Foi com o prof. Güntzel que consolidamos a atuação do GACI na UFPel e, deste modo, eu pude avançar nas atividades de investigação relacionadas ao meu doutorado enquanto ainda não havia sido liberado. Hoje, estou prestes a retornar às atividades docentes na UFPel e o vazio

deixado pelo prof. Güntzel tem ecoado insistentemente em minha mente e em meu coração.

João, Leandro, Fabiane, Rafael e André, meus bolsistas da UFPel, que trabalharam (e trabalham) duro, mesmo com seu orientador estando longe, merecem meu muito obrigado. Foram eles que mantiveram a produtividade nas minhas áreas de interesse junto ao nosso grupo de pesquisa mesmo quando eu estava afastado. Cada um deles contribuiu em alguma medida com meu doutorado e, espero, quando voltar para a UFPel, poder contribuir ainda mais com as suas formações.

Também sou muito grato a todos os funcionários do Instituto de Informática da UFRGS que, muitas vezes de forma anônima, contribuem para que o instituto funcione tão bem. Em especial, gostaria de agradecer à Zíngara e à Jane, que incontáveis vezes me ajudaram nas confusões das minhas viagens e na gerência dos projetos em que estou envolvido. Além delas, não posso deixar de citar a Beatriz, a Elisiane e o Luis Otávio, que estão sempre disponíveis para resolver os problemas dos usuários do Instituto.

Nesta caminhada, entre muitos contatos e encontros, tive a felicidade de conhecer o Prof. Clóvis Tondo, da *Florida Atlantic University*. Este brasileiro e gaúcho, radicado nos Estados Unidos há várias décadas, é um exemplo de cidadania. Professor, autor de diversos livros, empresário de sucesso, piloto de avião, pára-quedista, mergulhador, faixa preta de caratê, e mesmo com todas as conquistas e compromissos, não esqueceu que nasceu, cresceu e aprendeu no interior do Rio Grande do Sul. O prof. Tondo doou diversos livros para o GACI, auxiliando na construção de minha tese e na orientação do pessoal envolvido com as pesquisas que desenvolvemos. Além disso, já perdemos as contas de quantos livros foram doados para a UFPel pelo prof. Tondo, auxiliando fortemente na qualificação de nosso acervo. Por tudo isso e por todos os resultados positivos que ainda vamos colher juntos, sou muito grato ao prof. Tondo.

Não poderia deixar de agradecer a duas pessoas especiais em minha vida e que, infelizmente, no decorrer desta jornada, acabaram ficando pelo caminho. São elas minha avó Hilda e minha avó adotada Oraídes. Lembro de inúmeras lições da vó Hilda que foram e são importantes na minha vida. De todas elas, porém, as mais importantes são o carinho e o amor com que cativava a todos ao seu redor. Da dona Oraídes, que perdemos recentemente, eu sinto falta da presença, do carinho, da atenção e do cuidado. Ela era um pilar firme onde sempre podíamos amarrar nossas angústias e preocupações. As duas se foram deixando enormes saudades, mas, acima de tudo, deixando muitas e importantes lições de vida que irão me acompanhar pelo resto dos meus dias.

A todos os Volcan, com quem partilhei minha vida desde a infância, sou muito grato. É incrível, mas a enorme maior parte dos melhores momentos da minha vida aconteceu sempre ao lado de um ou mais membros desta família. Meus tios, cada um de sua maneira, foram modelos que tenho tentado seguir em minha vida, em especial a tia Gicelda, a tia Hilda, o tio Gilberto, o tio Círio, a tia Gilca e o tio Luís Carlos. Luís Carlos não, tio Frank! A ele devo a ressurreição de muitas de minhas utopias, tão abaladas pelos eventos históricos recentes de nosso país. São poucos os que têm clareza para enxergar caminhos no meio das trevas e muito raros àqueles que têm coragem de apontar este caminho para outros sob os quais se abateu a cegueira permanente. Tio Frank, muito obrigado por tudo! Também tenho que agradecer a todos meus numerosos primos, em especial ao Vander, à Jô, ao Rogério e à Adri. É claro que não me esqueci de agradecer aos meus primos e irmãos: Rodrigo e Romi. Aos dois, meu muito obrigado por existirem e por terem compartilhado tantas experiências boas em más junto comigo.

Com os Agostini, em função da distância, tenho mantido um contato menos constante, embora muito importante. A eles, agradeço os exemplos de alegria, de acolhida e de amizade. Em especial, gostaria de agradecer à tia Maurília (minha madrinha), à tia Anastácia, ao tio Guerino e à tia Lurdes. Além destes, gostaria de destacar dois primos Agostini importantes para mim: o Marcos, que contribuiu muito para ampliar minhas perspectivas de mundo e o João Pedro, que sempre me inspirou com sua coragem, dedicação e luta frente ao MST.

Também gostaria de lembrar a todos aqueles que não foram citados e que, de uma forma ou outra contribuíram com minha caminhada. Estes foram muitos, tantos, que se eu citasse todos, utilizaria mais páginas nos agradecimentos do que em todo o resto da tese.

Finalmente, devo agradecer ao Vitor Ramil, que, cantando em milonga os versos de João da Cunha Vargas, conseguiu tocar meu coração, por descrever com grande precisão tudo o que sinto e o que sou. Como no poema, eu "fui criado solito" e "sempre gostei da morena" e sempre achei "linda a liberdade". Sou eu o "índio do queixo torcido que se amansou na experiência" e meu doutorado foi uma daquelas experiências que amansa qualquer vivente. Também eu "nunca pensei que minha sina fosse andar longe do pago", com tanto tempo vivendo em Porto Alegre e com tantas andanças pelo mundo. Faz quase sete anos que deixei "o rancho da infância coberto pela neblina" e confesso que, hoje, olho para trás e fico admirado pelo tanto que andei e, desta experiência, pude ter certeza que realmente "o mundo é pequeno". Mas agora, após esta defesa de tese, tenho a alegria de saber que todo o caminho que percorri me conduziu de volta para casa. Agora eu vou "voltar pra querência, lugar onde fui parido" com muita vontade de trabalhar e produzir e, acima de tudo, de ser feliz.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

1-D	Uma Dimensão
2-D	Duas Dimensões
AC	*Alternate Current*
ASO	*Arbitrary Slice Order*
AVC	*Advanced Video Coding*
B	*Bi-predictive*
Bck	*Block Subsampling*
bS	*Boundary Strength*
CABAC	*Context-Based Adaptive Binary Arithmetic Coding*
CAVLC	*Context-Based Adaptive Variable Length Coding*
Cb	*Chrominance blue*
CCIR	*Consultative Committee for International Radio*
CIF	*Common Intermediate Format*
codec	codificador/decodificador
Cr	*Chrominance red*
DC	*Direct Current*
DCS	*Dual Cross Search*
DCT	*Discrete Cosine Transform*
DPCM	*Differential Pulse Code Modulation*
DS	*Diamond Search*
DVD	*Digital Versatile Disk*
EDK	*Embedded Development Kit*
FDCT	*Forward Discrete Cosine Transform*
FIFO	*First In First Out*
FIR	*Finite Impulse Response*
FPGA	*Field Programmable Gate Array*
FRExt	*Fidelity Range Extensions*

FS	*Full Search*
H422P	*High 4:2:2 Profile*
H444P	*High 4:4:4 Profile*
HDTV	*High Definition Digital Television*
Hi10P	*High 10 Profile*
HP	*High Profile*
HS	*Hexagon Search*
HSI	*Hue, Saturation, Intensity*
I	*Inter*
I_PCM	*Intra-frame Pulse Code Modulation*
IDCT	*Inverse Discrete Cosine Transform*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electric and Electronics Engineers*
II	coluna ímpar, linha ímpar
IP	coluna ímpar, linha par
ISO	*International Organization for Standardization*
ITU-T	*International Telecommunication Union - Telecommunication*
JVT	*Joint Video Team*
LDSP	*Large Diamond Search Pattern*
MAE	*Mean Absolute Error*
MC	*Motion Compensation*
ME	*Motion Estimation*
MIPS	*Million Instructions Per Second*
MMCVM	Máximo Módulo da Componente do Vetor de Movimento
MP3	*MPEG-1 Audio Layer 3*
MPEG	*Moving Picture Experts Group*
MPVF	Módulo Prototipado em Verificação Funcional
MSB	*Most Significant Bit*
MSE	*Mean Square Error*
MV	*Motion Vector*
MVPr	*Motion Vector Prediction*
NAL	*Network Adaptation Layer*
OT	*One at a Time Search*
P	*Predictive*
PC	*Personal Computer*

PDA	*Personal Digital Assistant*
Pel	*Pel Subsampling*
PI	coluna par, linha ímpar
POC	*Picture Order Count*
PP	coluna par, linha par
PSNR	*Peak Signal-to-Noise Ratio*
Q	*Quantization*
Q^{-1}	*Inverse Quantization*
QCIF	*Quarter Common Intermediate Format*
QP	*Quantization Parameter*
Qstep	*Quantization Step*
RAM	*Random Access Memory*
RGB	*Red, Green, Blue*
RLB	Registrador de Linha de Busca
RLE	*Run Length Encoding*
RLP	Registrador de Linha de Pesquisa
SAD	*Sum of Absolute Differences*
SAE	*Sum of Absolute Errors*
SDRAM	*Synchronous Dynamic Random Access Memory*
SDSP	*Small Diamond Search Pattern*
SDTV	*Standard Definition Television*
SI	*Switching I*
SP	*Switching P*
T	*Transform*
T^{-1}	*Inverse Transform*
TSS	*Three Step Search*
TV	Televisão
UFPel	Universidade Federal de Pelotas
UFRGS	Universidade Federal do Rio Grande do Sul
UFRN	Universidade Federal do Rio Grande do Norte
UP	Unidade de Processamento
V2P	*Virtex 2 Pro*
VCEG	*Video Coding Experts Group*
VCL	*Video Coding Layer*
VGA	*Video Graphics Array*

VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
VLC	*Variable Length Coding*
VLSI	*Very Large Scale Integration*
XUP	*Xilinx University Program*
Y	*Luminance*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A compressão de vídeo é essencial para aplicações que manipulam vídeos digitais, em função da enorme quantidade de informação necessária para representar um vídeo sem nenhum tipo de compressão. Esta tese apresenta o desenvolvimento de soluções arquiteturais dedicadas e de alto desempenho para a compressão de vídeos, com foco no padrão H.264/AVC. O padrão H.264/AVC é o mais novo padrão de compressão de vídeo da ITU-T e da ISO e atinge as mais elevadas taxas de compressão dentre todos os padrões de codificação de vídeo existentes. Este padrão também possui a maior complexidade computacional dentre os padrões atuais.

Esta tese apresenta soluções arquiteturais para os módulos da estimação de movimento, da compensação de movimento, das transformadas diretas e inversas e da quantização direta e inversa. Inicialmente, são apresentados alguns conceitos básicos de compressão de vídeo e uma introdução ao padrão H.264/AVC, para embasar as explicações das soluções arquiteturais desenvolvidas. Então, as arquiteturas desenvolvidas para os módulos das transformadas diretas e inversas, da quantização direta e inversa, da estimação de movimento e da compensação de movimento são apresentadas.

Todas as arquiteturas desenvolvidas foram descritas em VHDL e foram mapeadas para FPGAs Virtex-II Pro da Xilinx. Alguns dos módulos foram, também, sintetizados para *standard-cells*. Os resultados obtidos através da síntese destas arquiteturas são apresentados e discutidos. Para todos os casos, os resultados de síntese indicaram que as arquiteturas desenvolvidas estão aptas para atender as demandas de codecs H.264/AVC direcionados para vídeos de alta resolução.

Palavras-Chave: Codificação de Vídeo, Padrão H.264/AVC, Arquiteturas VLSI.

Design of High Performance Architectures Dedicated to Video Compression According to the H.264/AVC Standard

ABSTRACT

Video coding is essential for applications based in digital videos, given the enormous amount of bits which are required to represent a video sequence without compression. This thesis presents the design of dedicated and high performance architectures for video compression, focusing in the H.264/AVC standard. The H.264/AVC standard is the latest ITU-T and ISO standard for video compression and it reaches the highest compression rates amongst all the current video coding standards. This standard has also the highest computational complexity among all of them.

This thesis presents architectural solutions for the modules of motion estimation, motion compensation, forward and inverse transforms and forward and inverse quantization. Some concepts of video compression and an introduction to the H.264/AVC standard are presented and they serve as basis for the architectural developments. Then, the designed architectures for forward and inverse transforms, forward and inverse quantization, motion estimation and motion compensation are presented.

All designed architectures were described in VHDL and they were mapped to Xilinx Virtex-II Pro FPGAs. Some modules were also synthesized into standard-cells. The synthesis results are presented and discussed. For all cases, the synthesis results indicated that the architectures developed in this work are able to meet the demands of H.264/AVC codecs targeting high resolution videos.

Keywords: Video Coding, H.264/AVC Standard, VLSI Architectures.

1 INTRODUÇÃO

"Alcei a perna no pingo
E saí sem rumo certo"

João da Cunha Vargas

A compressão de vídeos digitais é um assunto bastante explorado na atualidade tanto pela academia, quanto pela indústria, dada a possibilidade de codificação e decodificação deste tipo de mídia por diversos dos utilitários eletrônicos atuais, como computador pessoal e portátil, aparelho celular, televisão digital de alta resolução (HDTV), DVD *players*, câmeras e filmadoras digitais portáteis, entre muitos outros. Em função do crescente avanço do interesse da indústria em codecs (codificadores/decodificadores) de vídeo, vários conjuntos de diferentes algoritmos e técnicas foram agregados para a criação de diversos padrões, visando o intercâmbio de vídeos digitais padronizados e a conseqüente potencialização da exploração comercial de todos os produtos relacionados. Aplicações em software e hardware foram desenvolvidas seguindo estes padrões, com o intuito de responder às restrições da aplicação alvo no que diz respeito à taxa de compressão, à qualidade da imagem, à flexibilidade, à taxa de processamento, ao consumo de energia, etc.

A compressão de vídeos é essencial para o sucesso das aplicações que manipulam vídeos digitais, pois um vídeo não comprimido utiliza uma quantidade de bits muito elevada. Isso implica em custos muito elevados em termos de armazenamento e de transmissão destas informações e estes custos acabam por dificultar o desenvolvimento de produtos para esta área, caso a compressão não seja utilizada. Por exemplo, considerando vídeos com resolução de 720x480 pixels a 30 quadros por segundo (usado em televisão digital com definição normal – SDTV e em DVDs), utilizando 24 bits por pixel, a taxa necessária para a transmissão sem compressão seria próxima a 249 milhões de bits por segundo (249 Mbps). Para armazenar uma seqüência de curta duração, com 10 minutos, seriam necessários quase 19 bilhões de bytes (19GB). Para vídeos com resolução de 1920x1080 pixels a 30 quadros por segundo (usado em televisão digital com alta definição ou HDTV), com 24 bits por pixel, a taxa de transmissão sobe para 1,5 bilhões de bits por segundo (1,5 Gbps) e seriam necessários 112 bilhões de bytes (112 GB) para armazenar um vídeo com 10 minutos.

Apesar das seqüências de vídeos digitalizados precisarem desta enorme quantidade de informação para serem representadas, estas seqüências possuem, em geral, uma outra importante propriedade intrínseca: apresentam elevado grau de redundância. Isto significa que uma boa parte da enorme quantidade de dados necessários para representar o vídeo digitalizado é desnecessária. O objetivo da compressão de vídeo é, justamente, o

desenvolvimento de técnicas que possibilitem a máxima eliminação possível destes dados desnecessários para, deste modo, representar o vídeo digital com um número de bits muito menor do que o original.

O padrão de compressão de vídeo H.264/AVC, foco deste trabalho, é o mais novo padrão de compressão de vídeo e foi desenvolvido com o objetivo de dobrar a taxa de compressão em relação aos demais padrões existentes até então. O padrão H.264/AVC foi desenvolvido pelo JVT (ITU-T, 2007a), que foi formado a partir de uma união entre os especialistas do VCEG da ITU-T (ITU-T, 2007) e do MPEG da ISO/IEC (ISO/IEC, 2007). A primeira versão do H.264/AVC foi aprovada em 2003.

O padrão H.264/AVC atingiu seu objetivo de alcançar as mais elevadas taxas de processamento dentre todos os padrões existentes, mas para isso foi necessário um grande aumento na complexidade computacional das operações dos codecs que seguem este padrão, em relação aos demais padrões disponíveis na atualidade. Este aumento de complexidade dificulta, pelo menos na tecnologia atual, a utilização de codecs H.264/AVC completos implementados em software, quando as resoluções são elevadas e quando se deseja tempo real, com 30 quadros por segundo, por exemplo. A dificuldade de tratar o problema via software, somada ao enorme interesse comercial que reside neste padrão, têm impulsionado equipes de pesquisa e desenvolvimento ao redor do mundo a tratarem deste tema visando otimizações algorítmicas e/ou implementações em hardware para que os requisitos das aplicações sejam atendidos. Além disso, outro forte motivador para o desenvolvimento de codecs H.264/AVC em hardware está na crescente necessidade de sistemas embarcados capazes de manipular vídeos. São exemplos as câmaras e filmadoras digitais, os PDAs, os MP3 *players*, os celulares, os *set top boxes* de TV digital, etc. Neste caso, questões como consumo de energia e quantidade de recursos de hardware utilizados são críticas. Para estas aplicações, a única alternativa é o desenvolvimento destes sistemas em hardware dedicado, pois um software rodando em um processador de propósito geral não atende aos requisitos típicos de sistemas embarcados para aplicações computacionalmente muito complexas, como é o caso dos codecs de vídeo.

Existem muitas aplicações potenciais para codecs H.264/AVC, que vão de celulares à televisão digital e, por isso, a indústria está extremamente ativa nesta área e algumas soluções para HDTV já estão disponíveis, principalmente para decodificadores (que são mais simples). Estas soluções comerciais costumam conter muitos segredos industriais, de modo que nenhuma destas soluções está reportada em detalhes na literatura. Do ponto de vista da academia, existem muitas equipes espalhadas pelo mundo trabalhando com o H.264/AVC, buscando soluções de software e/ou hardware para atacar o problema da complexidade elevada do padrão. Vários trabalhos têm sido publicados nos últimos anos, mas a área encontra-se ainda repleta de problemas sem solução e, consequentemente, muitas contribuições inovadoras podem ser descobertas e implementadas. Esta tese tem por objetivo contribuir com esta área de conhecimento, propondo algumas soluções inovadoras em hardware para este padrão.

A taxa de compressão do H.264/AVC varia de acordo com a qualidade pretendida para o vídeo codificado. Com uma taxa de 50:1, a redução na qualidade do vídeo é, em geral, pequena e pouco perceptível (ou mesmo imperceptível) ao sistema visual humano. Neste caso, o vídeo gerado possui uma qualidade equivalente a vídeos MPEG-2 usados em DVDs. Para uma primeira avaliação dos ganhos em compressão do codificador H.264/AVC em relação a vídeos não comprimidos, serão usados os mesmos exemplos citados anteriormente, ou seja, vídeos com resolução de 720x480 (SDTV) e

1920x1080 (HDTV) pixels a 30 quadros por segundo e utilizando 24 bits por pixel. Além disso, a taxa de compressão de 50:1 será tomada como referência. Para SDTV, a taxa de transmissão necessária com o uso do H.264/AVC é de cerca de 2,5 Mbps, enquanto um vídeo não comprimido usaria 249 Mbps. Os 10 minutos de vídeo SDTV, que ocupariam quase 19GB no vídeo não comprimido passam a ocupar apenas 380KB com o uso do H.264/AVC. Para vídeos HDTV, a taxa de transmissão necessária ao H.264/AVC fica abaixo de 30 Mbps, contra os 1,5 Gbps necessários sem o uso de compressão. A seqüência de 10 minutos passaria a ocupar 2,2MB ao invés dos 112 GB necessários ao vídeo sem compressão.

A partir da primeira análise do padrão, e da constatação de sua elevada complexidade computacional, surgiu uma instigante e inquietante questão: Será possível desenvolver soluções em hardware com as limitações da tecnologia atual para o padrão H.264/AVC capazes de atingir tempo real para vídeos de alta definição? Esta questão definiu o norte das investigações apresentadas nesta tese e, no decorrer do texto, ela será gradativamente respondida.

Nesta direção, o trabalho apresentado nesta tese teve o objetivo de investigar o padrão H.264/AVC e de construir soluções arquiteturais inovadoras de alto desempenho para os módulos de codecs H.264/AVC. Os módulos da estimação de movimento, da compensação de movimento, das transformadas diretas e inversas e da quantização direta e inversa serão os focos de desenvolvimento das arquiteturas apresentadas nesta tese. Os módulos da predição intra-quadro e do filtro redutor de efeito de bloco, presentes em codecs H.264/AVC, foram desenvolvidos em trabalhos paralelos no grupo de pesquisa em TV Digital da UFRGS e não serão tratados nesta tese. Algumas soluções para o módulo da codificação de entropia foram desenvolvidas no escopo deste trabalho, mas como estão em estágio inicial de desenvolvimento, não serão relatadas nesta tese.

Este trabalho está inserido no esforço acadêmico para construir o sistema brasileiro de TV digital, através da proposta da construção do codec H.264/AVC. Alguns resultados deste esforço geraram publicações coletivas que incluem resultados apresentados nesta tese, entre elas, as mais importantes foram (AGOSTINI, 2007a; ROSA, 2007; AGOSTINI, 2006a).

As soluções arquiteturais propostas foram desenvolvidas em VHDL, mapeadas para FPGAs Xilinx (XILINX, 2007), validadas e prototipadas em placas de teste. A prototipação foi direcionada para FPGAs, mas FPGAs não são a tecnologia mais adequada para implementação comercial deste tipo de aplicação, uma vez que os codecs H.264/AVC para o sistema brasileiro de TV digital estarão presentes em vários milhões de aparelhos de TV em nosso país. Neste caso, tanto pela quantidade de unidades comercializadas, quanto por algumas outras restrições como consumo de energia e facilidade de integração, a solução mais apropriada seria o desenvolvimento de um circuito integrado dedicado (ASIC) para este fim. Os resultados obtidos com a prototipação foram animadores e indicam que as arquiteturas desenvolvidas neste trabalho poderiam ser inseridas nos ASICs para codecs H.264/AVC da televisão digital brasileira.

A estrutura do texto desta tese está organizada como segue. O capítulo dois apresenta alguns conceitos básicos de compressão de vídeo, que serão úteis no decorrer do texto e que servem para contextualizar o trabalho. O capítulo três apresenta, resumidamente, o padrão H.264/AVC, seu histórico, sua terminologia, sua estrutura e

alguns detalhes sobre os principais módulos formadores de codecs H.264/AVC. Além disso, o capítulo três apresenta uma comparação com padrões anteriores, uma análise de complexidade computacional e uma discussão sobre os principais desafios relacionados à implementação deste padrão. O capítulo quatro apresenta as arquiteturas desenvolvidas para os módulos das transformadas e quantização, enquanto que o capítulo cinco apresenta as arquiteturas desenvolvidas para a predição inter-quadros (estimação e compensação de movimento). Por fim, o capítulo seis apresenta as conclusões desta tese.

2 CONCEITOS BÁSICOS DA COMPRESSÃO DE VÍDEO

"E a trotezito no mais.
Fui aumentando a distância"

João da Cunha Vargas

Este capítulo apresentará, de forma resumida, os conceitos básicos que serão importantes para a compreensão das arquiteturas tratadas nos demais capítulos do texto.

2.1 Espaço de Cores e Subamostragem de Cores

A representação digital de um vídeo colorido está associada à interpretação das cores pelo sistema visual humano. O sistema humano de visão possui elementos sensíveis à luz chamados bastonetes e cones. Os bastonetes são utilizados para visão noturna e apresentam um elevado ganho, mas captam imagens com baixa resolução, ou seja, com pouco nível de detalhes. Por outro lado, os cones são sensíveis às cores vermelho, verde, azul e amarelo e são capazes de identificar detalhes de elevada resolução, mas precisam de maior luminosidade para seu funcionamento. (POLLACK, 2006). As saídas dos cones são pré-processadas por camadas de neurônios presentes na própria retina, logo acima da dos cones. Esta camada de neurônios codifica a imagem em um canal de luminância (tons de cinza) e dois canais de crominância, um para gerenciar vermelho e verde e outro para gerenciar azul e amarelo (POLLACK, 2006). O sistema visual humano é capaz de discernir milhares de cores distintas a partir de combinações de intensidades diferentes das cores captadas pelos cones. Por outro lado, o sistema visual humano consegue distinguir mais do que cinco dúzias de tons de cinza, que indicam a intensidade luminosa da imagem (luminância) (GONZALEZ, 2003).

Existem muitas formas de se representar as cores de forma digital. Um sistema para representar cores é chamado de espaço de cores e a definição do espaço de cor a ser utilizado para representar um vídeo é essencial para a eficiência da codificação deste vídeo.

São vários os espaços de cores usados para representar imagens digitais, tais como: RGB, HSI e YCbCr (SHI, 1999). O espaço de cores RGB é um dos mais comuns e conhecidos, tendo em vista que é este o espaço de cores utilizado nos monitores coloridos para interação computador-usuário. O RGB representa, em três matrizes distintas, as três cores primárias captadas pelo sistema visual humano: vermelho, verde e azul. Daí é que surge o nome deste espaço de cores (do inglês *red*, *green*, *blue* – RGB). No espaço de cores YCbCr, as três componentes utilizadas são luminância (Y),

que define a intensidade luminosa ou o brilho; crominância azul (Cb) e crominância vermelha (Cr) (BHASKARAN, 1997).

Os componentes R, G e B possuem um elevado grau de correlação, o que não é desejável do ponto de vista da compressão de vídeos. Por isso, a compressão é aplicada para espaços de cores do tipo luminância e crominância, como o YCbCr (RICHARDSON, 2002).

Outra vantagem do espaço de cor YCbCr sobre o espaço RGB é que, no espaço YCbCr, a informação de cor está completamente separada da informação de brilho. Deste modo, estas informações podem ser tratadas de forma diferenciada pelos codificadores de imagens estáticas e de vídeos.

O sistema visual humano é mais sensível a informações de luminância do que a informações de crominância. Então, os padrões de compressão de imagens estáticas e vídeos podem explorar esta característica humana para aumentar a eficiência de codificação através da redução da taxa de amostragem dos componentes de crominância em relação aos componentes de luminância (RICHARDSON, 2002). Esta operação é chamada de subamostragem de cores e é realizada sob o espaço de cores YCbCr nos padrões de compressão de vídeo atuais.

Existem várias formas de relacionar os componentes de crominância com o componente de luminância para realizar a subamostragem. Os formatos mais comuns são o 4:4:4, o 4:2:2 e o 4:2:0. No formato 4:4:4, para cada quatro amostras de luminância (Y), existem quatro amostras de crominância azul (Cb) e quatro amostras de crominância vermelha (Cr). Por isso, os três componentes de cor possuem a mesma resolução e existe uma amostra de cada elemento de cor para cada pixel da imagem e, assim, a subamostragem não é aplicada. No formato 4:2:2, para cada quatro amostras de Y na direção horizontal, existem apenas duas amostras de Cb e duas amostras de Cr. Neste caso, as amostras de crominância possuem a mesma resolução vertical das amostras de luminância, mas possuem metade da resolução horizontal. No formato 4:2:0, para cada quatro amostras de Y, existe apenas uma amostra de Cb e uma amostra de Cr. Neste caso, as amostras de crominância possuem metade da resolução horizontal e metade da resolução vertical do que as amostras de luminância. A nomenclatura 4:2:0 é usada por motivos históricos, pois os números não representam a relação lógica entre os componentes de cor, que seria 4:1:1 (RICHARDSON, 2003).

A subamostragem de cor aumenta significativamente a eficiência da codificação, uma vez que parte da informação da imagem é simplesmente descartada, sem causar impacto visual perceptível. Considerando o formato 4:2:0 como exemplo, uma vez que cada componente de crominância possui exatamente um quarto das amostras presentes no componente de luminância, então um vídeo YCbCr no formato 4:2:0 irá utilizar exatamente a metade das amostras necessárias para um vídeo RGB ou YCbCr no formato 4:4:4. Isso implica em uma taxa de compressão de 50%, considerando apenas a subamostragem.

O padrão H.264/AVC, foco deste trabalho, considera que os dados do vídeo de entrada estão no espaço de cores YCbCr. Subamostragens de cor nos formatos 4:2:0, 4:2:2 e 4:4:4 são permitidas, mas o formato mais usado é o 4:2:0. Este formato é muito adequado para a capacidade de percepção do sistema visual humano.

2.2 Redundância de Dados na Representação de Vídeos

A codificação de vídeos busca diminuir a quantidade de dados considerados redundantes na representação computacional das informações da imagem ou do vídeo. Considera-se redundante aquele dado que não contribui com novas informações relevantes para a representação da imagem. Basicamente, existem três tipos diferentes de redundâncias exploradas na compressão de vídeos: redundância espacial, redundância temporal e redundância entrópica. Cada uma destas redundâncias será resumidamente explicada nos próximos parágrafos.

- **Redundância Espacial** – A redundância espacial é, também, chamada de "redundância intra-quadro" (GHANBARI, 2003) ou "redundância interpixel" e advém da correlação existente entre os pixels espacialmente distribuídos em um quadro. Esta correlação pode ser percebida, tanto no domínio espacial, quanto no domínio das freqüências. Esta correlação é visualmente percebida no domínio espacial quando são observados pixels vizinhos em um quadro, que tendem a possuir valores semelhantes. Neste caso, a redundância pode ser reduzida através da operação chamada de codificação "intra-quadro", presente em alguns padrões de codificação de vídeo atuais. No domínio das freqüências a operação realizada para reduzir a redundância espacial é chamada de quantização. Para aplicar a quantização, antes as informações da imagem devem ser transformadas do domínio espacial para o domínio das freqüências. A quantização é uma divisão inteira dos coeficientes gerados pela transformação e reduz grande parte dos coeficientes à zero. Esta operação é irreversível, pois o resto da divisão não é armazenado e, deste modo, a quantização gera perdas no processo de codificação. Mas é importante ressaltar que estas perdas tendem a interferir de forma nula ou pouco significativa na qualidade perceptual da imagem.
- **Redundância Temporal** – A redundância temporal, também chamada de "redundância inter-quadros" (GHANBARI, 2003), é causada pela correlação existente entre quadros temporalmente próximos em um vídeo. Na verdade, a redundância temporal poderia ser classificada como apenas mais uma dimensão da redundância espacial, como faz (GONZALEZ, 2003). Muitos blocos de pixels simplesmente não mudam de valor de um quadro para outro em um vídeo, como por exemplo, em um fundo que não foi alterado de um quadro para outro. Outros pixels apresentam uma pequena variação de valores causada, por exemplo, por uma variação de iluminação. Por fim, também é possível que o bloco de pixels simplesmente tenha se deslocado de um quadro para o outro, como por exemplo, em um movimento de um objeto em uma cena. Todos os padrões atuais de codificação de vídeo visam reduzir a redundância temporal. A exploração eficiente da redundância temporal conduz a elevadas taxas de compressão, o que é fundamental para o sucesso dos codificadores.
- **Redundância Entrópica** – A redundância entrópica está relacionada com as probabilidades de ocorrência dos símbolos codificados. A entropia é uma medida da quantidade média de informação transmitida por símbolo do vídeo (SHI, 1999). A quantidade de informação nova transmitida por um símbolo diminui na medida em que a probabilidade de ocorrência deste símbolo aumenta. Então, os codificadores que exploram a redundância entrópica têm por objetivo transmitir o máximo de informação possível por símbolo codificado e, deste modo, representar mais informações com um número menor de bits. A "codificação de

entropia", como é chamada, utiliza diferentes técnicas e algoritmos de compressão sem perdas para atingir este objetivo.

2.3 Modelo de Codificador/Decodificador de Vídeo

Antes de entrar em maiores detalhes sobre o padrão H.264/AVC propriamente dito, será definido um modelo simplificado de codificador e de decodificador de vídeo, com base na eliminação das redundâncias relacionadas na seção anterior. Este modelo tem como base os padrões atuais de compressão de vídeo e tem o intuito de localizar, com mais clareza, cada uma das principais operações realizadas em um codificador ou decodificador de vídeo.

A Figura 2.1 apresenta o modelo inicial de codificador. Existem, neste modelo simplificado, dois quadros do vídeo sendo utilizados simultaneamente. Além do quadro atual, que está sendo comprimido, também é utilizado um quadro de referência anteriormente processado. Os quadros são divididos em diversas partes, normalmente chamadas de blocos, para serem processados.

Para os modelos apresentados nesta seção, são considerados apenas dois tipos de quadros. Os quadros do tipo I são quadros utilizados no início do vídeo e para sincronizar o sinal de vídeo. Os quadros tipo I podem ser codificados apenas com blocos do tipo I. A codificação dos blocos I não depende de blocos de quadros anteriores e é realizada considerando apenas as informações contidas no quadro. Deste modo, nos quadros do tipo I é explorada a redundância espacial. Os quadros tipo P são os quadros mais utilizados e são construídos a partir do quadro atual e de quadros previamente codificados. Os quadros do tipo P podem ser codificados usando blocos tipo I ou tipo P. Os blocos tipo P são blocos codificados a partir de blocos de quadros já processados e, por isso, exploram a redundância temporal. Então, um quadro do tipo P pode explorar, simultaneamente, as redundâncias espacial e temporal, dependendo do tipo de blocos que são usados na sua codificação.

Os blocos do quadro atual são codificados através da codificação inter-quadros ou através da codificação intra-quadro. A chave seletora na Figura 2.1 representa a decisão tomada pelo hardware de controle do codificador sobre qual modo de codificação deve ser utilizado para cada bloco.

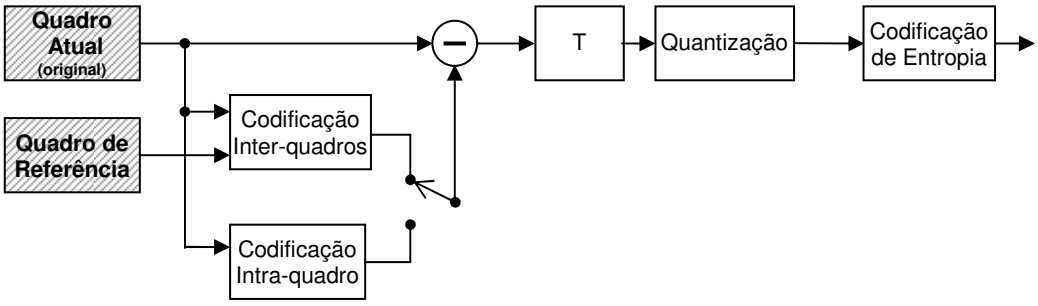

Figura 2.1: Modelo inicial de codificador de vídeo

O módulo de codificação intra-quadro é responsável por reduzir a redundância espacial, utilizando, para tanto, apenas a informação do quadro atual em processamento. Este módulo é responsável por gerar blocos tipo I, usados na codificação de quadros tipo I ou P. Vários algoritmos podem ser utilizados para realizar a codificação intra-

quadro e estes algoritmos são os mesmos utilizados em compressão de imagens estáticas. Os algoritmos de codificação intra-quadro de interesse para este trabalho serão apresentados no próximo capítulo.

O módulo de codificação inter-quadros é responsável por reduzir a redundância temporal, através da comparação dos blocos do quadro atual com os blocos do quadro de referência. Este módulo gera blocos do tipo P, para quadros do tipo P. O bloco do quadro de referência com maior semelhança ao bloco que está sendo processado do quadro atual é escolhido. Então, um vetor de movimento é gerado para identificar a posição deste bloco no quadro de referência. A predição inter-quadros é, normalmente, dividida em duas etapas chamadas de estimação de movimento e compensação de movimento. O termo "estimação de movimento", neste contexto, não seria o mais apropriado, uma vez que o termo "estimativa de movimento" seria mais correto. Como o jargão da área utiliza amplamente o termo "estimação de movimento", iremos utilizar este termo nesta tese.

Após um bloco do quadro atual ser codificado pela codificação intra-quadro ou pela codificação inter-quadros, é realizada uma subtração entre os valores do bloco original e os resultados da codificação. Esta diferença é chamada de resíduo.

O resíduo, então, é enviado para os módulos responsáveis por reduzir a redundância espacial no domínio das freqüências. A primeira operação nesta direção é a transformada (módulo T na Figura 2.1). O módulo T transforma a informação do domínio espacial para o domínio das freqüências. Neste domínio, a Quantização pode ser aplicada, reduzindo a redundância espacial presente nos resíduos.

Por fim, a codificação de entropia reduz a redundância entrópica, que está relacionada à forma como os dados são codificados. Novamente, diversos algoritmos podem ser empregados para este fim e alguns destes algoritmos, de interesse para este trabalho, serão apresentados no próximo capítulo desta tese.

O modelo apresentado na Figura 2.1 atua na redução dos três tipos de redundância apresentados na seção anterior. Mas este modelo ainda não está completo e não é útil para a codificação de vídeo. É possível observar que o codificador não armazenou o resultado codificado do quadro atual. Deste modo, quando um próximo quadro é processado, o quadro atual não codificado passa a ser o quadro de referência. Como apenas o quadro codificado é transmitido, o decodificador não receberá o quadro original, recebendo apenas o quadro codificado. Como a codificação gera perdas de informação, o quadro codificado será diferente do quadro original após a decodificação (RICHARDSON, 2003). Então, as referências utilizadas pelo codificador e pelo decodificador serão diferentes para o processamento de um mesmo quadro. Esta diferença na referência utilizada inviabiliza a aplicação deste tipo de solução. Para resolver este problema, o codificador descarta o quadro original depois de ser processado, e armazena o quadro reconstruído. A informação reconstruída é relevante tanto para a codificação inter-quadros quanto para a codificação intra-quadro. Na codificação inter-quadros, o quadro codificado é usado como quadro de referência para a codificação do próximo quadro. A codificação intra-quadro, em alguns padrões, utiliza as amostras dos blocos vizinhos que acabaram de ser codificados, como referência para a codificação dos próximos blocos. Em ambos os casos, a referência do codificador e do decodificador devem ser as mesmas. Para isso ser possível, parte da tarefa da decodificação é inserida também no codificador.

A Figura 2.2 apresenta o diagrama de blocos mais detalhado do codificador, agora considerando a reconstrução da imagem.

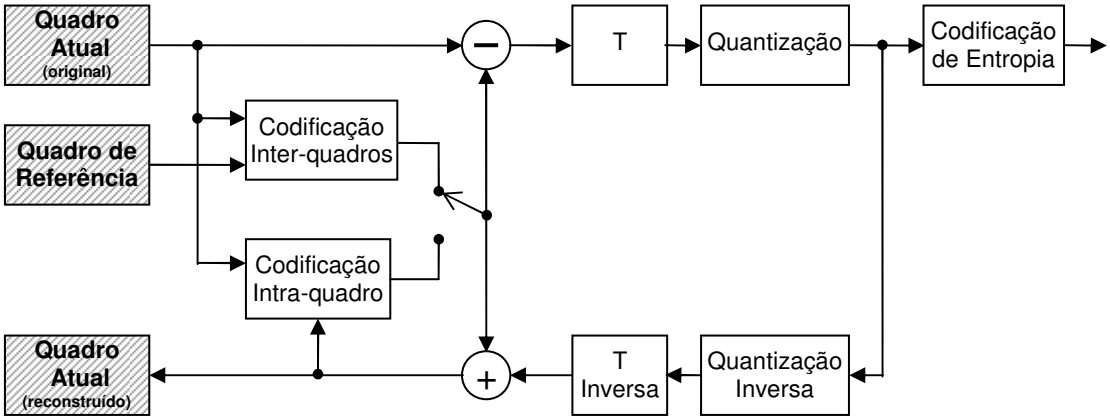

Figura 2.2: Modelo completo de codificador de vídeo

Como as perdas de informação, que geram a diferença entre o quadro atual original e o quadro atual reconstruído, acontecem no estágio de quantização, a imagem deve ser reconstruída a partir deste ponto. A codificação e decodificação de entropia são desnecessárias para a reconstrução do quadro atual justamente porque estas etapas não geram perdas. Então, é aplicada a operação inversa da quantização e, após a quantização inversa, é aplicada a transformada inversa, gerando os resíduos reconstruídos. Aos resíduos é somado o resultado da codificação intra-quadro ou inter-quadros do bloco reconstruído. Finalmente, o bloco reconstruído está pronto e pode ser armazenado para ser utilizado pela codificação inter-quadros do próximo quadro ou pode ser utilizado diretamente pela codificação intra-quadro dos próximos blocos do quadro atual.

Finalmente, o modelo de codificador apresentado na Figura 2.2 está completo. Este tipo de codificador de vídeo é chamado de codificador híbrido, por explorar diversos tipos de redundância (BHASKARAN, 1997).

O modelo genérico do decodificador está apresentado na Figura 2.3. Como pode ser observado a partir de uma comparação da Figura 2.3 com a Figura 2.2, a decodificação é muito parecida com a parte da reconstrução do quadro da codificação.

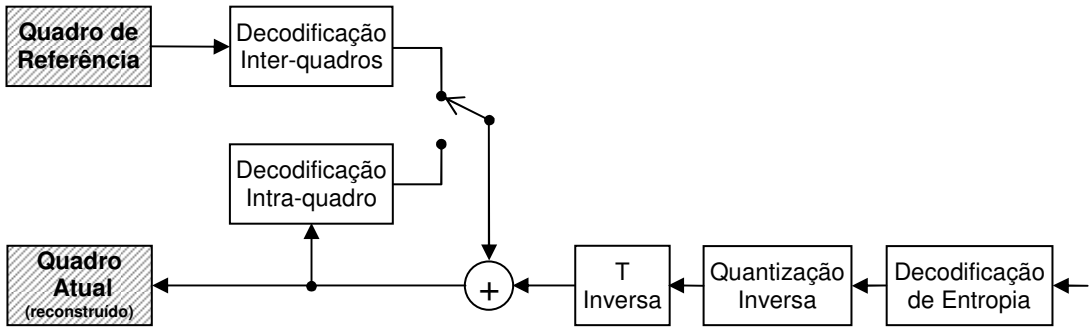

Figura 2.3: Modelo de decodificador de vídeo

A decodificação de entropia reconstrói as amostras codificadas. Como a codificação de entropia não gera perdas de informação, a operação de decodificação de entropia gera exatamente o mesmo resultado gerado pela saída da quantização no codificador.

Após a decodificação de entropia, as amostras remontadas são entregues para a quantização inversa e para a transformada inversa, reconstruindo o resíduo. Estas operações são idênticas às apresentadas no codificador.

A decodificação inter-quadros utiliza o quadro de referência e as informações de controle (não apresentadas na Figura 2.3), para localizar os blocos dentro do quadro de referência. Estes blocos são, então, somados aos resíduos, para gerar os blocos reconstruídos. Finalmente, os blocos reconstruídos são armazenados para serem usados como referência para a decodificação do próximo quadro e para a visualização do vídeo decodificado.

A decodificação intra-quadro utiliza os blocos reconstruídos do quadro atual e informações de controle (omitidas na Figura 2.3), para reconstruir os blocos codificados pelo codificador inter-quadros. De maneira similar à codificação inter-quadros, os macroblocos que passam pela decodificação intra-quadro são somados aos resíduos e armazenados para serem usados como referência e para visualização do vídeo.

É importante salientar que, nos modelos apresentados nas Figuras 2.2 e 2.3, não estão apresentados os fluxos de sinais de controle.

2.4 Métricas de Comparação

Existem muitas técnicas e algoritmos bastante distintos para comprimir vídeos e os padrões de compressão de vídeo da atualidade exploram estas possibilidades de formas variadas. Para comparar os resultados obtidos, é importante definir alguns critérios de comparação.

O primeiro critério de comparação é a eficiência de codificação, que é definida como sendo a redução no volume de dados obtidos por um determinado algoritmo de compressão (SANBORN, 2005). Por exemplo, uma eficiência de compressão de 70% significa que somente 30% dos dados originais são usados após a codificação.

Mas além da eficiência de codificação, é muito importante definir critérios para comparar a qualidade das imagens geradas. A medição da qualidade visual para determinar a qualidade de um algoritmo de codificação de vídeos é uma tarefa bastante imprecisa, porque há muitos fatores que influenciam nesta definição e muitos destes fatores são inerentemente subjetivos. Por isso, existem dois tipos principais de métricas para avaliar qualidade de um vídeo (RICHARDSON, 2003): métricas subjetivas e métricas objetivas. A qualidade subjetiva não será discutida nesta tese, pois não foi utilizada nas avaliações apresentadas.

A qualidade objetiva é medida usando um algoritmo que compara o vídeo original com o vídeo codificado. Esta comparação é realizada quadro a quadro, comparando todos os pixels dos quadros do vídeo original com os pixels do vídeo codificado.

O PSNR (*Peak Signal-to-Noise Ratio*) é o parâmetro de avaliação mais utilizado para comparar a qualidade objetiva de vídeos (BHASKARAN, 1997) e está definido em (1). O PSNR é usado para realizar uma comparação entre dois quadros distintos que, neste caso, são o quadro original e o quadro reconstruído. Em (1), MAX é o valor máximo que uma amostra de luminância do vídeo pode atingir. O PSNR é medido em uma escala logarítmica, utilizando, como base, o critério de similaridade MSE (*Mean Squared Error*) entre o quadro original e o quadro reconstruído. O MSE será definido nos próximos parágrafos,

$$PSNR = 20 \times \log\left(\frac{MAX}{\sqrt{MSE}}\right) \qquad (1)$$

A qualidade objetiva pode ser diretamente ligada ao valor do PSNR obtido. Valores altos indicam alta qualidade e valores baixos de PSNR indicam baixa qualidade. É importante salientar que o PSNR possui uma série de limitações como critério de qualidade, sendo que, em alguns casos, uma imagem que possui uma qualidade visual (subjetiva) superior pode ter um valor de PSNR inferior à outra com um resultado visual pior (RICHARDSON, 2003). Isto ocorre porque o PSNR considera apenas o valor do MSE para cada pixel da imagem e, desta forma, não necessariamente avalia os critérios subjetivos de qualidade (RICHARDSON, 2003).

Uma medida importante para avaliar a diferença entre dois quadros é o grau de similaridade entre eles. Este grau de similaridade pode ser mensurado a partir de diferentes funções de similaridade. As duas funções de similaridade de interesse para este trabalho são o MSE, usado na definição do PNSR, e o SAD (*Sum of Absolute Differences*), usado na predição inter-quadros.

O cálculo do MSE está definido em (2), onde m e n são as dimensões do quadro a ser comparado e O e R são as amostras dos quadros original e reconstruído, respectivamente.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (R_{i,j} - O_{i,j})^2 \qquad (2)$$

O SAD calcula a distorção entre as regiões comparadas, a partir do somatório das diferenças absolutas, para cada ponto do quadro original e do quadro reconstruído (KUHN, 1999). A função para o cálculo do SAD é dada por (3). Novamente, m e n são as dimensões do quadro a ser comparado e O e R são as amostras dos quadros original e reconstruído, respectivamente.

$$SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left| R_{i,j} - O_{i,j} \right| \qquad (3)$$

3 O PADRÃO H.264/AVC

"Eu fui criado solito
Mas sempre bem prevenido
Índio do queixo torcido
Que se amansou na experiência"

João da Cunha Vargas

O padrão H.264/AVC (ITU-T, 2005), foco desta tese, é o mais novo padrão de codificação de vídeo e foi desenvolvido pelo JVT (*Joint Video Team*) (ITU-T, 2007a). O JVT é formado por especialistas do VCEG (*Video Coding Experts Group*) da ITU-T (ITU-T, 2007) e do MPEG (*Motion Photographic Experts Group*) da ISO (ISO/IEC, 2007). Este capítulo da tese apresenta maiores detalhes sobre o padrão H.264/AVC, desde seu histórico, até detalhes sobre os seus principais módulos.

3.1 Histórico do Padrão

O padrão H.264/AVC foi desenvolvido por um período de aproximadamente quatro anos. As raízes deste padrão estão no projeto H.26L da ITU-T. O H.26L foi iniciado pelo VCEG no início de 1998, através da construção da chamada para propostas. O primeiro *draft* deste novo padrão foi finalizado em agosto de 1999. O objetivo do projeto H.26L era dobrar a eficiência de codificação atingida pelo padrão H.263 (ITU-T, 2000).

Antes do padrão H.264/AVC surgir, alguns outros padrões já haviam sido criados e já estavam consolidados, servindo de base para o desenvolvimento do H.264/AVC. O primeiro padrão relevante para a construção do H.264/AVC foi o H.261 da ITU-T (ITU-T, 1990). Este padrão lançou as bases do que é usado até hoje na maioria dos padrões de compressão de vídeo: DPCM com estimação de movimento na direção temporal, transformada discreta do cosseno aplicada no resíduo e quantização linear seguida de codificação de entropia. Após o padrão H.261, surgiu o padrão MPEG-1 da ISO/IEC (ISO/IEC, 1993), seguido do padrão MPEG-2 da ISO/IEC, que também foi padronizado pela ITU-T como H.262 (ITU-T, 1994). O MPEG-2 ou H.262 se tornou um padrão popular e é muito utilizado até a atualidade em diversas aplicações. Apesar do grande sucesso do padrão MPEG-2, a evolução dos padrões de compressão de vídeo não parou. O padrão H.263 (ITU-T, 2000) foi lançado e incorporou alguns avanços obtidos pelos padrões MPEG-1 e MPEG-2, bem como técnicas novas que vinham sendo pesquisadas intensamente tanto pela indústria quanto pela academia.

Em 2001, o grupo MPEG da ISO/IEC (ISO/IEC, 2007) finalizou o desenvolvimento do padrão MPEG-4 Parte 2 (ISO/IEC, 1999). Ainda neste ano, o MPEG construiu uma nova chamada de propostas, similar à do H.26L da ITU-T, para melhorar ainda mais a eficiência de codificação atingida pelo MPEG-4. Então, o VCEG, da ITU-T resolveu submeter seu *draft* em resposta à chamada de propostas do MPEG e propôs a união de esforços para completar o trabalho.

Analisando as respostas para sua chamada de propostas, o MPEG chegou a conclusões que reafirmaram as escolhas de desenvolvimento realizadas pelo VCEG no H.26L (SULLIVAN, 2004):

- A estrutura de compensação de movimento com a transformada discreta do cosseno (DCT) era melhor do que as outras.
- Algumas ferramentas de codificação de vídeo que foram excluídas no passado (do MPEG-2, do H.263 ou do MPEG-4 Parte 2) por causa da sua complexidade computacional, poderiam ser reexaminadas para inclusão no próximo padrão, devido aos avanços na tecnologia de hardware.
- Para maximizar a eficiência de codificação, a sintaxe do novo padrão não poderia ser compatível com os padrões anteriores.

Para permitir um avanço mais acelerado na construção do novo padrão e para evitar duplicação de esforços, o ITU-T e o ISO/IEC concordaram em unir esforços para desenvolverem, em conjunto, a próxima geração de padrão para codificação de vídeo e concordaram em usar o H.26L como ponto de partida. Então, foi criado, em dezembro de 2001, o JVT (*Joint Video Team*) (ITU-T, 2007a), formado por especialistas do VCEG e do MPEG. O JVT tinha o objetivo de completar o desenvolvimento técnico do padrão até o ano de 2003. A ITU-T decidiu adotar o padrão com o nome de H.264 e a ISO/IEC decidiu adotar o padrão com o nome de MPEG-4 parte 10 – AVC (*Advanced Video Coding* - Codificação de Vídeo Avançada). O padrão H.264 teve seu *draft* final (ITU-T, 2003) aprovado em outubro de 2003 (SULLIVAN, 2005).

Em um artigo de 2004, descrevendo o padrão (SULLIVAN, 2004), o coordenador do JVT, Gary Sullivan, decidiu utilizar o nome H.264/AVC para se referir ao padrão de forma equilibrada entre os nomes dados ao padrão pela ITU-T e pela ISO. O H.264 foi retirado do nome dado ao padrão pela ITU-T, enquanto que o AVC foi extraído do nome do padrão na ISO. Este nome, com um misto dos nomes oficiais do padrão, acabou sendo usado por muitos autores desde então e, deste ponto em diante neste texto, o padrão sempre será referenciado desta forma.

Em julho de 2004, o JVT adicionou algumas novas funcionalidades ao padrão H.264/AVC através de uma extensão do padrão chamada de *Fidelity Range Extensions* (FREext) (ITU-T, 2004; ITU-T, 2005).

3.2 Terminologia

Para o padrão H.264/AVC a formação de uma imagem codificada pode acontecer a partir de um campo ou quadro, quando o vídeo é entrelaçado, ou a partir de um quadro, quando o vídeo é progressivo (RICHARDSON, 2003).

O padrão H.264/AVC considera que o espaço de cores utilizado para representar o vídeo de entrada é do tipo luminância e crominância, como o YCbCr. A relação entre os elementos Y, Cb e Cr é dependente do perfil do padrão que está sendo considerado. Esta relação e os perfis do padrão serão apresentados ainda nesta seção.

Um quadro codificado é formado por um número determinado de macroblocos, cada um contendo 16x16 amostras de luminância, associadas às amostras de crominância. Dependendo do perfil utilizado, o tamanho das amostras de crominância de um macrobloco pode variar, como ficará mais claro nos próximos parágrafos. Além das amostras da imagem presentes nos macroblocos, o quadro codificado também contém informações de controle, que indicam o tipo de codificação adotado, o início de macrobloco, o tipo de macrobloco, etc. Cada elemento codificado, seja com informações de controle ou com amostras da imagem, é chamado de elemento sintático (RICHARDSON, 2003).

Um quadro codificado possui um número de quadro, que é sinalizado no *bitstream*. Este número de quadro não está, necessariamente, relacionado à ordem de decodificação deste quadro. Cada campo codificado de um quadro progressivo ou entrelaçado possui um contador que indica a sua ordem de decodificação. Os quadros codificados anteriormente podem ser utilizados como quadros de referência para a predição de outros quadros. Os quadros de referência são organizados em duas listas, chamadas de lista 0 e lista 1.

Dentro de cada quadro, os macroblocos são organizados em *slices*, onde um *slice* é um grupo de macroblocos em uma ordem de varredura tipo *raster*. Um *slice* do tipo I pode conter somente macroblocos do tipo I. Um *slice* do tipo P pode conter macroblocos do tipo P e I e um *slice* do tipo B pode conter macroblocos do tipo B e I. Além dos *slices* tipo I, P e B, o padrão também permite a existência de outros dois tipos de *slices*: SI e SP (ITU-T, 2005). Os slices SI e SP não serão detalhados neste texto porque não fazem parte do foco desta tese. Um quadro do vídeo pode ser codificado em um ou mais *slices*, cada um contendo um número inteiro de macroblocos. O número de macroblocos pode variar de um até o número total de macroblocos do quadro (RICHARDSON, 2003).

Os macroblocos do tipo I são codificados usando a codificação intra-quadro, a partir das amostras do *slice* atual. A codificação pode acontecer sobre macroblocos completos ou para cada bloco. Os macroblocos do tipo I também podem ser codificados através do modo I_PCM, onde o codificador transmite os valores das amostras diretamente, sem predição ou transformação. Em alguns casos anômalos, o modo I_PCM pode ser mais eficiente do que os demais modos de codificação (RICHARDSON, 2003).

Os macroblocos do tipo P são codificados usando a codificação inter-quadros, a partir de quadros de referência previamente codificados. Um macrobloco codificado no modo inter-quadros pode ser dividido em partições de macroblocos, isto é, em blocos de 16x16, 16x8, 8x16 ou 8x8. Se o tamanho de partição escolhido for o 8x8, então cada sub-macrobloco 8x8 pode ser dividido novamente em partições de sub-macrobloco de tamanho 8x8, 8x4, 4x8 ou 4x4. Cada partição de macrobloco é codificada utilizando como referência um quadro da lista 0. Se existir uma partição de sub-macrobloco, então esta partição é codificada utilizando o mesmo quadro da lista 0 utilizado para codificar a partição de macrobloco.

Os macroblocos do tipo B também são codificados usando a codificação inter-quadros. Cada partição de macrobloco pode ser codificada utilizando um ou dois

quadros de referência, um na lista 0 e outro na lista 1. Se existir uma partição em sub-macrobloco, cada sub-macrobloco é codificado a partir dos mesmos um ou dois quadros de referência utilizados na codificação da partição de macrobloco.

3.3 Perfis e Níveis

O padrão H.264/AVC é dividido em diferentes perfis. Cada perfil suporta um grupo particular de funções de codificação e especifica o que é necessário para cada codificador e decodificador que seguem este perfil. A primeira versão do padrão H.264/AVC, de maio de 2003 (ITU-T, 2003), define um grupo de três diferentes perfis: *Baseline*, *Main* e *Extended*. O perfil *Baseline* é direcionado a aplicações como vídeo-telefonia, videoconferência e vídeo sem fio. O perfil *Baseline* suporta codificação intra-quadro e inter-quadros (usando somente *slices* I e P) e uma codificação de entropia com códigos de comprimento de palavra variável adaptativos ao contexto (CAVLC). O perfil *Main* é focado na transmissão de televisão e no armazenamento de vídeo. O perfil *Main* inclui o suporte para vídeo entrelaçado, o suporte à codificação inter-quadros utilizando *slices* do tipo B e utilizando predição ponderada e o suporte à codificação de entropia utilizando codificação aritmética adaptativa ao contexto (CABAC). O perfil *Extended* é mais voltado para aplicações em *streaming* de vídeo e não suporta vídeo entrelaçado ou o CABAC, mas agrega modos para habilitar uma troca eficiente entre *bitstreams* codificados (através de *slices* do tipo SP e SI) e melhora a resiliência a erros (através do particionamento de dados).

Para os três perfis definidos na primeira versão do padrão, a relação entre os elementos de cores é fixa. Esta relação é de 4:2:0 para os elementos Y, Cb e Cr. Isso significa que os elementos de crominância Cb e Cr possuem metade da resolução horizontal e vertical dos elementos de luminância. Isso implica em uma subamostragem dos elementos de crominância já no vídeo original. Cada macrobloco, como já foi apresentado anteriormente, é formado por uma região de 16x16 pixels de um quadro do vídeo, incluindo as informações de luminância e crominância. Com a relação de 4:2:0, um macrobloco é formado por uma matriz de 16x16 amostras de luminância e por duas matrizes 8x8 amostras de crominância, uma para Cb e outra para Cr.

Os perfis *Baseline*, *Main* e *Extended* também possuem outra característica em comum: nos três perfis são usados 8 bits por amostra.

Estes três perfis inicialmente propostos pelo padrão H.264/AVC não incluíram suporte para vídeos com qualidade mais elevada, como as necessárias em ambientes profissionais. Para responder às exigências deste tipo de aplicação, uma continuação do projeto JVT foi realizada para adicionar novas extensões para as capacidades do padrão original. Estas extensões foram chamadas de extensões para alcance de fidelidade (*fidelity range extensions* - FRExt). O FRExt produziu um grupo de quatro novos perfis chamados coletivamente de perfis *High* (SULLIVAN, 2004).

Os perfis *High* possuem algumas características comuns que são inovadoras em relação aos perfis originais: suportam um tamanho de bloco adaptativo para a transformada (4x4 ou 8x8), suportam matrizes de quantização baseadas em percepção e suportam uma representação sem perdas de regiões específicas do conteúdo do vídeo (SULLIVAN, 2004).

As matrizes de quantização baseadas em percepção foram usadas anteriormente no padrão MPEG-2. Com esta ferramenta, o codificador pode especificar, para cada

tamanho de bloco da transformada e separadamente para codificação intra-quadro e inter-quadros, um fator de escala específico. Isso permite um ajuste da fidelidade da quantização de acordo com o modelo de sensibilidade do sistema visual humano para diferentes tipos de erros. Tipicamente, esta ferramenta não melhora a fidelidade objetiva (PSNR), mas melhora a fidelidade subjetiva, que é o critério realmente mais importante.

O perfil *High* (HP) inclui vídeo com 8 bits por amostra e com relação de cor 4:2:0. O perfil *High* 10 (Hi10P) suporta vídeo a 10 bits por amostra, também com uma relação de cor 4:2:0. O perfil *High* 4:2:2 (H422P) inclui suporte à relação de cor 4:2:2 e vídeos a 10 bits por amostra. Finalmente, o perfil *High* 4:4:4 (H444P) dá suporte à relação de cor 4:4:4 (ou seja, sem nenhuma subamostragem), suporte a vídeo com até 12 bits por amostra e, adicionalmente, suporta a codificação sem perdas em regiões do vídeo.

A Figura 3.1 apresenta, resumidamente, a relação entre os perfis *Baseline*, *Main*, *Extended* e *High* do padrão H.264/AVC.

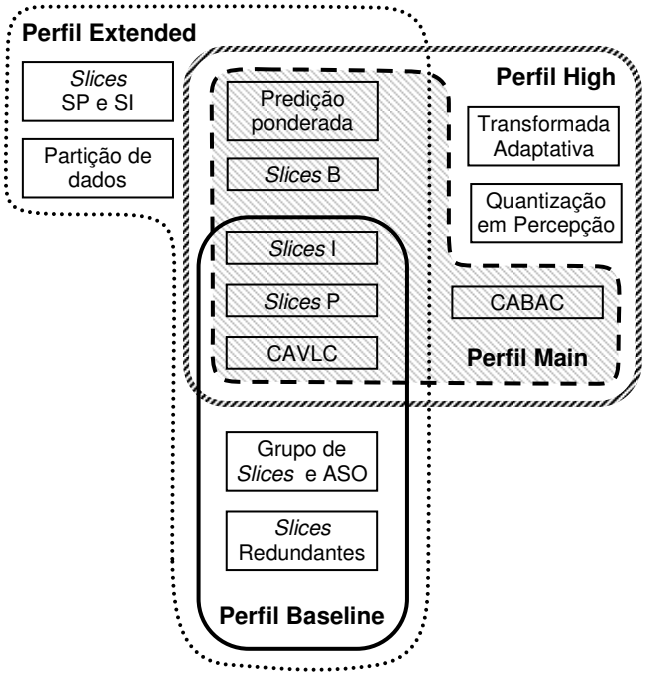

Figura 3.1: Perfis *Baseline*, *Main*, *Extended* e *High* do padrão H.264/AVC

Além da divisão em diversos perfis, o padrão H.264/AVC também define 16 diferentes níveis em função da taxa de processamento e da quantidade de memória necessária para cada implementação. Com a definição do nível utilizado, é possível deduzir o número máximo de quadros de referência e a máxima taxa de bits que podem ser utilizados (SULLIVAN, 2004).

3.4 Formato de Dados Codificados

Uma interessante inovação do padrão H.264/AVC é que ele faz uma clara distinção entre a organização do vídeo codificado e da sua transmissão via rede. O padrão classifica a informação do vídeo em dois diferentes níveis de abstração, chamados de camada de vídeo codificado (VCL – *Vídeo Coding Layer*) e camada de abstração de rede (NAL – *Network Abstration Layer*) (RICHARDSON, 2003). Os dados de saída do processo de codificação estão na camada VCL, sendo formados por uma seqüência de bits que representam os dados do vídeo codificado. Estes dados são mapeados para

unidades NAL antes da transmissão ou armazenamento. Uma seqüência de vídeo codificado é representada por uma seqüência de unidades NAL que podem ser transmitidas sobre uma rede baseada em pacotes, podem ser transmitidas via um link de transmissão de *bitstream* ou ainda, podem ser armazenados em um arquivo.

O objetivo de especificar separadamente a VCL e a NAL é diferenciar características específicas da codificação (na VCL) daquelas características específicas do transporte (na NAL). Neste trabalho não serão apresentados em maiores detalhes o mecanismo de transporte utilizado na NAL. Maiores detalhes podem ser encontrados na literatura (ITU-T, 2005; RICHARDSON,2003).

3.5 Comparação com Padrões Anteriores

A Tabela 3.1 apresenta uma comparação entre os padrões H.264/AVC, MPEG-4 Parte 2 e MPEG-2 do ponto de vista das diferentes características suportadas por estes padrões (KWON, 2005). As características do H.264/AVC que ainda não foram discutidas nas seções anteriores, serão explicadas em mais detalhes no decorrer do texto.

Tabela 3.1: Comparação das características dos padrões H.264/AVC, MPEG-4 Parte 2 e MPEG-2

	H.264/AVC	MEPG-4 parte 2	MPEG-2
Tamanho de Macrobloco	16x16	16x16	16x16 (modo quadro) 16x8 (modo campo)
Tamanho de bloco	16x16, 8x16, 16x8, 8x8, 4x8, 8x4, 4x4	16x16, 16x8, 8x8	8x8
Predição Intra-quadro	Domínio espacial	Domínio das freqüências	Não
Transformada	DCT inteira 8x8, 4x4 Hadamard 4x4, 2x2	DCT 8x8 / Wavelet	DCT 8x8
Codificação de Entropia	VLC, CAVLC, CABAC	VLC	VLC
Precisão	¼ pixel	¼ pixel	½ pixel
Quadros de referência	Múltiplos quadros	Um quadro	Um quadro
Modo de predição bidirecional	Para frente / para trás Para frente / para frente Para trás / para trás	Para frente / para trás	Para frente / para trás
Predição ponderada	Sim	Não	Não
Filtro de blocagem	Sim	Não	Não
Tipos de quadros	I, P, B, SI, SP	I, P, B	I, P, B
Perfis	7 perfis	8 perfis	5 perfis
Taxa de transmissão	64Kbps a 150Mbps	64Kbps a 2Mbps	2Mbps a 15Mbps
Complexidade do codificador	Elevada	Média	Média

Fonte: KWON, 2005.

O padrão H.264/AVC teve como principal objetivo, como já foi citado anteriormente, aumentar a eficiência da codificação em relação aos padrões anteriores. Nos próximos parágrafos será apresentada uma comparação entre o padrão H.264/AVC e outros padrões, com foco na eficiência de codificação. Os dados apresentados nesta seção foram colhidos da literatura e apresentam algumas discrepâncias, mas todos indicam que o padrão H.264/AVC apresenta ganhos significativos sobre outros padrões em termos de eficiência de codificação.

O trabalho de Kamaci (2003) apresenta uma comparação entre o padrão H.264/AVC, o padrão MPEG-2 e o padrão H.263, este último nos perfis *Baseline* e *High*. Nesta comparação, é considerada a relação sinal/ruído de pico (PSNR) sobre vídeos nos formatos ITU-R-601, CIF e QCIF. De acordo com os autores, o padrão H.264/AVC atinge cerca 50% de ganho de codificação sobre o padrão MPEG-2, cerca de 47% de ganho sobre o padrão H.263 *Baseline* e cerca de 24% de ganho sobre o padrão H.263 *High*. Isso significa que, para um mesmo PSNR, o padrão H.264/AVC atinge uma taxa de compressão significativamente mais elevadas do que os padrões anteriores.

Wiegand (2003) compara a relação sinal ruído entre os padrões MPEG-2, H.263, MPEG-4 parte 2 e H.264/AVC. Foram usadas seqüências de vídeos no formato QCIF e CIF, com diferentes taxas de quadros por segundo. A Tabela 3.2 apresenta os resultados comparativos nos ganhos médios na taxa de bits do padrão H.264/AVC sobre os demais padrões.

Tabela 3.2: Ganhos comparativos na taxa de bits do padrão H.264/AVC em relação a outros padrões

Aplicação	MPEG-4 Parte 2 (%)	H.263 (%)	MPEG-2 (%)
Streaming	37,4	47,6	63,6
Vídeo Conferência	29,4	40,6	-
Entretenimento / Qualidade	-	-	45

Fonte: WIEGAND, 2003.

As Figuras 3.2 e 3.3 apresentam exemplos de gráficos comparativos obtidos a partir dos diversos experimentos realizados por Wiegand (2003). Nestas figuras são apresentados os gráficos do PSNR para diferentes taxas de bits e do ganho de compressão em relação ao MPEG-2. A Figura 3.2 apresenta a comparação quando a aplicação de *streaming* é considerada. A Figura 3.3 apresenta a comparação para aplicações de entretenimento.

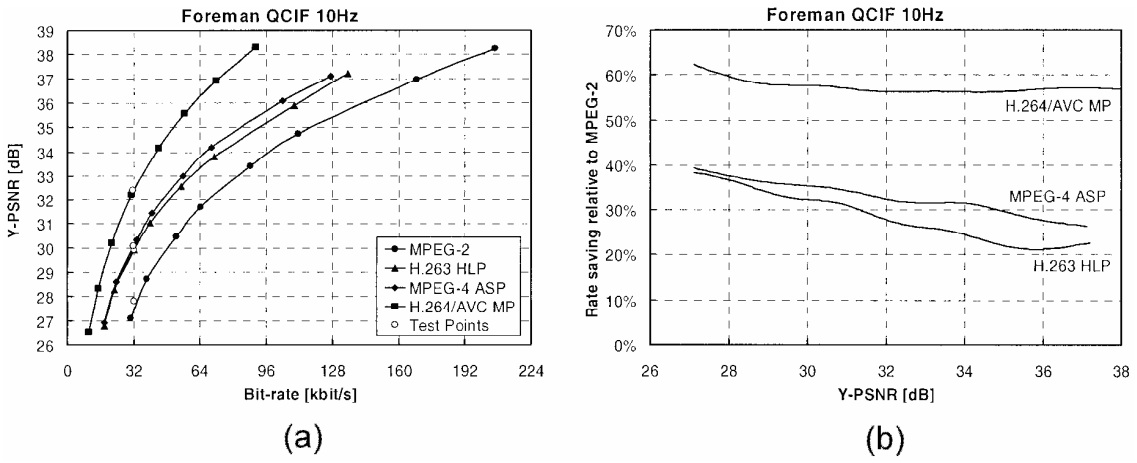

Figura 3.2: Comparação (a) do PSNR para diferentes taxas de bits e (b) ganho de compressão em relação ao MPEG-2, para o vídeo "Foreman" considerando resolução de vídeo para *streaming* (WIEGAND, 2003)

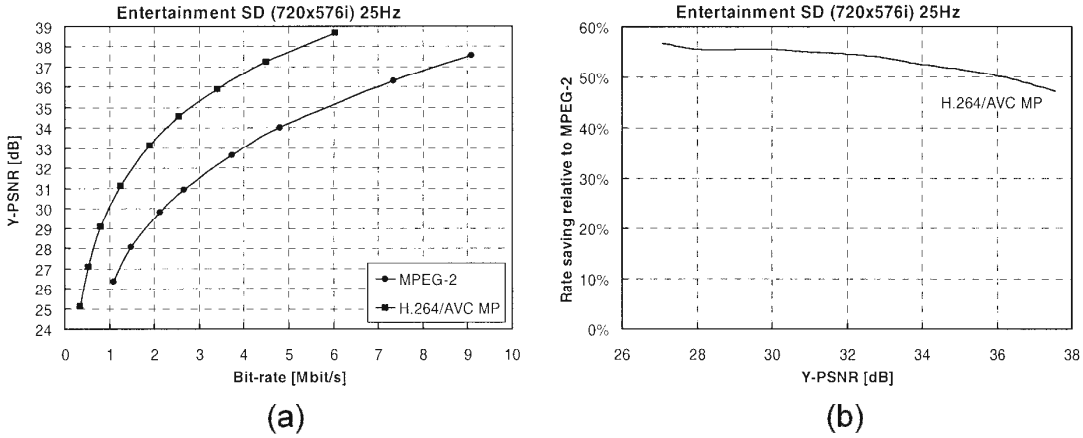

Figura 3.3: Comparação (a) do PSNR para diferentes taxas de bits e (b) ganho de compressão em relação ao MPEG-2, para o vídeo "Entertainment" considerando resolução de vídeo para entretenimento (WIEGAND, 2003)

Em todos os casos apresentados nas Figuras 3.2, 3.3 e 3.4, o padrão H.264/AVC apresentou desempenho superior aos demais padrões. Esta capacidade do padrão H.264/AVC de atingir as maiores eficiências de codificação para diversos tipos de aplicação é uma das mais importantes vantagens deste padrão em relação aos demais padrões de compressão de vídeo que, normalmente, são direcionados para um tipo de aplicação específica.

Os ganhos na relação sinal ruído vêm acompanhados de um considerável aumento da complexidade computacional dos algoritmos utilizados no H.264/AVC em relação aos algoritmos utilizados nos padrões anteriores. Segundo Sunna (2005) a relação entre o acréscimo na complexidade versus os ganhos na eficiência da codificação, quando o padrão MPEG-2 é tomado como base, é definido de acordo com os dados apresentados na Tabela 3.3. Na comparação da Tabela 3.3, são considerados os três perfis presentes na primeira versão do padrão H.264/AVC e é considerada a complexidade apenas do decodificador H.264/AVC. Os dados apresentados na Tabela 3.3 são aproximados e apresentam apenas uma idéia dos ganhos de eficiência e do acréscimo em complexidade.

Tabela 3.3: Relação entre o H.264/AVC e o MPEG-2 quanto aos ganhos em eficiência de codificação e o acréscimo de complexidade.

Perfil	Ganho em Eficiência de Codificação (vezes)	Acréscimo de Complexidade (vezes)
Baseline	1,5	2,5
Extended	1,75	3,5
Main	2	4

Fonte: SUNNA, 2005.

O acréscimo na complexidade computacional de um codificador H.264/AVC é ainda mais elevado do que o acréscimo na complexidade do decodificador quando o padrão MPEG-2 é utilizado como comparação. Considerando o codificador, o H.264/AVC é cerca de oito vezes mais complexo do que o MPEG-2.

3.6 Núcleo do Codec H.264/AVC

Esta seção do texto irá apresentar os principais módulos de um codificador e de um decodificador H.264/AVC, bem como suas funcionalidades.

O diagrama de blocos do codificador H.264/AVC está apresentado na Figura 3.4. É possível perceber que a Figura 3.4 é muito parecida com aquela apresentada na seção 2.3 desta proposta. Aquele modelo apresentado na Figura 2.2 foi justamente construído para servir de base para discussão que será apresentada nesta seção do texto sobre o compressor H.264/AVC.

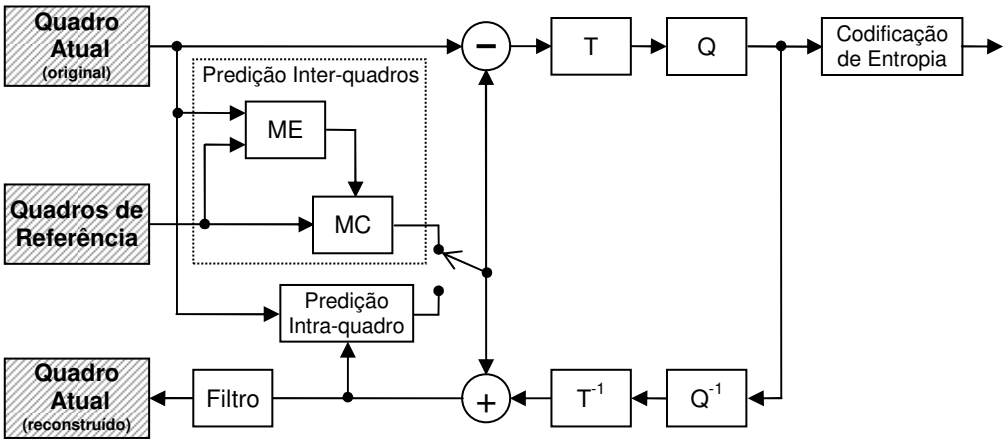

Figura 3.4: Diagrama de blocos de um codificador H.264/AVC

O módulo de predição inter-quadros na Figura 3.4 é formado pela estimação de movimento (ME – *Motion Estimation*) e pela compensação de movimento (MC – *Motion Compensation*). Este módulo desempenha função similar a do módulo de codificação inter-quadros da Figura 2.2, mas, para desempenhar esta função, o padrão H.264/AVC agrega operações de grande complexidade computacional a este módulo.

O módulo de predição intra-quadro, os módulos das transformadas diretas e inversas (T e T^{-1}), os módulos da quantização direta e inversa (Q e Q^{-1}) e o módulo da Codificação de Entropia na Figura 3.4 possuem a mesma função dos módulos com os mesmos nomes na Figura 2.2.

A única diferença mais significativa dentre a Figura 3.4 e a Figura 2.2 está no módulo chamado de Filtro na Figura 3.4, que não está presente na Figura 2.2. Este módulo foi inserido no padrão H.264/AVC e é requisito obrigatório. Nos demais padrões de compressão de vídeo este filtro não é obrigatório, mas um filtro similar é usado na prática por várias implementações seguindo outros padrões. Este filtro, também chamado de filtro de deblocagem ou filtro de *loop*, é usado para minimizar o efeito de bloco. Este filtro será explicado em maiores detalhes na seção 3.6.8 deste texto.

A Figura 3.5 apresenta o diagrama em blocos de um decodificador H.264/AVC. Novamente esta figura é muito parecida com a Figura 2.3, apresentada na seção 2.3. Também no decodificador H.264/AVC está presente o filtro, que não está originalmente apresentado na Figura 2.3.

Cada um dos módulos do codificador e do decodificador será explicado, nas próximas seções, de acordo com sua funcionalidade dentro do padrão H.264/AVC.

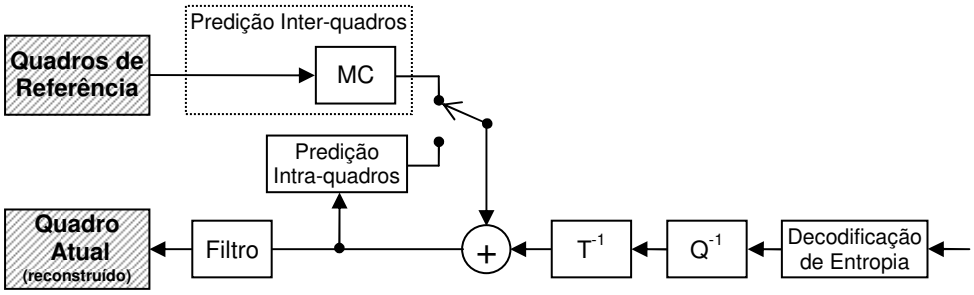

Figura 3.5: Diagrama de blocos de um decodificador H.264/AVC

Nas etapas de estimação e compensação de movimento (KUHN, 1999) do padrão H.264/AVC (bloco ME na Figura 3.4 e módulo MC nas Figuras 3.4 e 3.5) é onde se encontram a maior parte das inovações e dos ganhos obtidos pelo H.264/AVC sobre os demais padrões. Uma importante inovação é o uso de blocos de tamanho variável, onde se pode usar uma partição do macrobloco em blocos de tamanho 16x16, 16x8, 8x16, 8x8, 4x8, 8x4 ou 4x4 para realizar a ME e a MC. O H.264/AVC também prevê o uso de múltiplos quadros de referência, que não precisam ser somente os quadros I ou P imediatamente anteriores ou posteriores. Também é uma inovação o uso de novas opções para a bi-predição, como a predição ponderada e direta para os slices do tipo B. Além disso, os vetores de movimento podem apontar para fora da borda do quadro. Por fim, a predição de vetores de movimento com base nos vetores vizinhos também é uma novidade (ITU-T, 2005; RICHARDSON, 2003; WIEGAND, 2003a).

A etapa de predição intra-quadro (Figuras 3.4 e 3.5) é outra inovação introduzida pelo padrão H.264/AVC, pois é realizada no domínio espacial. Assim, mesmo nos macroblocos do tipo I é realizada uma predição antes da aplicação da transformada. Esta predição leva em conta as amostras já codificadas do *slice* atual.

No que diz respeito às transformadas direta e inversa (blocos T e T^{-1} nas Figuras 3.4 e 3.5), o H.264/AVC introduziu algumas inovações. A primeira delas é que as dimensões da transformada foram inicialmente definidas em 4x4 (ao invés do tradicional 8x8). Com a inserção dos perfis *High*, as transformadas passaram a ter dimensões variáveis de 4x4 ou 8x8 amostras. Outra inovação é relativa ao uso de uma aproximação inteira da transformada DCT direta e inversa, de modo a facilitar a sua implementação em hardware de ponto fixo e evitar erros de casamento entre o codificador e o decodificador (MALVAR, 2003). Além disso, uma segunda transformada é aplicada sobre os elementos DC resultantes da DCT para todos os blocos de crominância (Hadamard 2x2) e para os macroblocos em que é feita a predição intra-quadro 16x16 (Hadamard 4x4) (RICHARDSON, 2003).

A quantização no padrão H.264/AVC (blocos Q e Q^{-1} nas Figuras 3.4 e 3.5) é uma quantização escalar (RICHARDSON, 2002). O fator de quantização é função de um parâmetro **QP** de entrada, que é usado no codificador para controlar a qualidade da compressão e o *bit-rate* de saída.

O padrão H.264/AVC normatiza a utilização de um filtro redutor do efeito de bloco (módulo Filtro nas Figuras 3.4 e 3.5). Este filtro era opcional na maioria dos demais padrões de compressão de vídeo, mas passou a ser obrigatório no H.264/AVC. Uma inovação importante do filtro definido no padrão H.264/AVC é que ele é adaptativo, conseguindo distinguir entre uma aresta real da imagem de um artefato gerado por um elevado passo de quantização.

Na codificação de entropia (Figuras 3.4 e 3.5) o H.264/AVC também introduz ferramentas que aumentam bastante a eficiência de codificação deste módulo. Há, basicamente, dois tipos de codificação: a codificação de comprimento variável adaptativa ao contexto (CAVLC) e a codificação aritmética binária adaptativa ao contexto (CABAC). A principal inovação é o uso de codificação adaptativa baseada em contextos. Nesta codificação, a maneira com que os diversos elementos sintáticos são codificados depende do elemento a ser codificado, da fase em que se encontra o algoritmo de codificação e dos elementos sintáticos que já foram codificados.

Uma característica interessante do padrão H.264/AVC é que apenas o decodificador é normatizado. Então vários graus de liberdade podem ser explorados na implementação do codificador. O codificador deve gerar um *bitstream* compatível com as exigências do padrão, mas a forma como este *bitstream* é gerado não é normatizada. Então várias opções de implementação podem ser realizadas no codificador que vão deste a inserção de algoritmos mais eficientes ou mais fáceis de serem implementados em hardware, até a simples eliminação de algumas possibilidades de codificação previstas pelo padrão. É claro que estas modificações podem gerar impactos na relação sinal/ruído, na taxa de compressão, na velocidade de processamento e/ou na utilização de recursos de hardware pelo codificador e estes impactos devem ser criteriosamente avaliados para tomar qualquer decisão nesta direção.

3.6.1 O Módulo da Estimação de Movimento (ME)

A predição inter-quadros no codificador H.264/AVC, é formada pelos módulos da Estimação de Movimento (ME) e da Compensação de Movimento (MC). Esta seção focará o módulo da Estimação de Movimento, mas ambos os módulos estão estreitamente relacionados.

O módulo da estimação de movimento está presente apenas nos codificadores. Este módulo é o que apresenta a maior complexidade computacional dentre todos os módulos de um codificador H.264/AVC (PURI, 2004). Este grande custo computacional é função das inovações inseridas neste módulo do padrão, que tiveram o objetivo de atingir elevadas taxas de compressão. Residem nos módulos da ME e MC as principais fontes de ganhos do H.264/AVC em relação aos demais padrões de compressão de vídeo (WIEGAND, 2003, RICHARDSON, 2003).

A estimação de movimento deve prover as ferramentas de codificação capazes de localizar, nos quadros de referência, qual macrobloco mais se assemelha ao macrobloco atual. Assim que o macrobloco é encontrado, a ME deve gerar um vetor indicando a posição deste macrobloco no quadro de referência. Este vetor é chamado de vetor de movimento e deve ser inserido junto com a codificação do macrobloco.

Para a realização da estimação de movimento é considerado apenas o componente de luminância do macrobloco. Cada componente de crominância possui a metade da resolução horizontal e vertical do componente de luminância, então os componentes horizontal e vertical de cada vetor de movimento são divididos por dois para serem aplicados aos blocos de crominância.

A principal inovação do H.264/AVC no ponto de vista da estimação de movimento está na possibilidade de utilização de tamanhos de blocos variáveis para realizar a estimação de movimento. Ao invés de usar um macrobloco inteiro na estimação de movimento, o padrão H.264/AVC permite o uso de partições de macrobloco e partições de sub-macroblocos. As partições de macroblocos possuem tamanhos de 16x16, 8x16,

16x8 e 8x8, como está apresentado na Figura 3.6 (WIEGAND, 2003a, RICHARDSON, 2003).

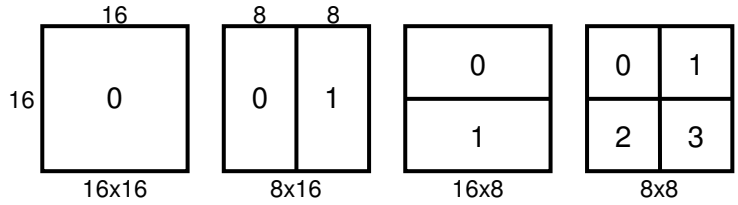

Figura 3.6: Divisão do macrobloco em partições de macroblocos

As partições de sub-macroblocos são permitidas somente se a partição de macrobloco selecionada foi a de 8x8. Neste caso, as quatro partições 8x8 do macrobloco podem ser divididas em mais quatro formas que podem ter o tamanho 8x8, 4x8, 8x4 ou 4x4, como está apresentado na Figura 3.7.

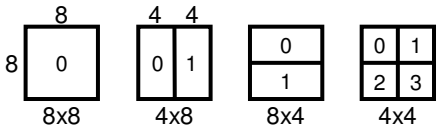

Figura 3.7: Divisão de uma partição de macrobloco em partições de sub-macroblocos

Este método de particionar macroblocos em sub-blocos de tamanho variável é conhecido como compensação de movimento estruturada em árvore.

Cada bloco de crominância é dividido da mesma maneira que os blocos de luminância, exceto pelo tamanho da partição, que terá exatamente a metade da resolução horizontal e vertical da partição de luminância. Por exemplo, uma partição de 8x16 de luminância corresponde a uma partição de 4x8 de crominância.

Um vetor de movimento é necessário para cada partição de macrobloco ou sub-macrobloco. Cada vetor de movimento precisa ser codificado e transmitido no *bitstream*, bem como o tipo de partição escolhida. A escolha de um tamanho de partição grande (16x16, 16x8, 8x16) implica na utilização de um número menor de vetores de movimento e também uma quantidade menor de bits são necessários para indicar o tipo de partição utilizada. Por outro lado, o resíduo gerado pode conter uma quantidade significativa de energia em áreas com muitos detalhes. A escolha de um tamanho de partição pequeno (8x4, 4x4) pode gerar um resíduo com quantidade de energia mínima depois da compensação de movimento, mas esta escolha requer a utilização de um número maior de vetores de movimento, além da necessidade de identificar a partição de sub-macrobloco escolhida. Assim é possível perceber que a escolha do tamanho da partição possui um impacto significativo no desempenho da compressão (RICHARDSON, 2003).

Em geral, uma partição grande é apropriada para áreas mais homogêneas do quadro e uma partição menor tende a ser mais apropriada para áreas com muitos detalhes.

O tamanho de partição é escolhido no algoritmo do codificador a partir de alguma métrica que conduza a uma codificação mais eficiente, isto é, que analise o compromisso entre a geração de um resíduo mínimo e a geração de um número mínimo de vetores de movimento. A melhor escolha é realizada tomando por base os resultados da análise para todos os tamanhos de partição de cada macrobloco. Então, é escolhido aquele tamanho de partição que apresentar a maior eficiência de codificação. Pode-se concluir que a complexidade computacional desta operação é bastante grande, uma vez

que os cálculos da estimação de movimento são realizados para vários tamanhos de partição diferentes e apenas uma destas partições é escolhida. Por isso, a estimação de movimento é o módulo mais crítico na implementação de codificadores H.264/AVC.

Uma outra característica importante da estimação de movimento do padrão H.264/AVC é que ela prevê uma precisão de ¼ de pixel para os vetores de movimento. Normalmente, os movimentos que acontecem de um quadro para o outro não estão restritos a posições inteiras de pixel. Assim, se são utilizados apenas vetores de movimento com valores inteiros, normalmente não é possível encontrar casamentos bons. Por isso, o padrão H.264/AVC prevê a utilização de vetores de movimento com valores fracionários de ½ pixel e de ¼ de pixel. Na Figura 3.8 é apresentado um exemplo da precisão do vetor de movimento com valores fracionários.

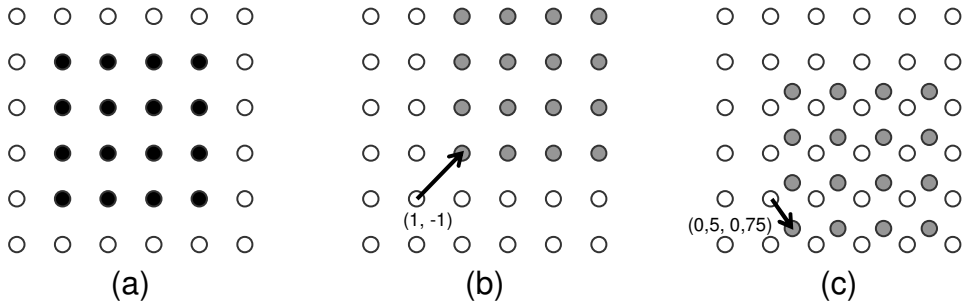

Figura 3.8: Estimação de movimento com precisão de frações de pixel

Na Figura 3.8 (a) está representado, com pontos pretos, um bloco 4x4 do quadro atual, que será predito a partir de uma região do quadro de referência na vizinhança da posição do bloco atual. Se os componentes horizontal e vertical do vetor de movimento são inteiros, então as amostras necessárias para o casamento estão diretamente presentes no quadro de referência. Como exemplo, a Figura 3.8 (b) apresenta, com pontos em cinza, um possível casamento do bloco atual no quadro de referência, onde o vetor de movimento utiliza apenas valores inteiros. Neste caso o vetor de movimento é (-1, 1). Se um ou os dois componentes do vetor de movimento apresentam valores fracionários, então as amostras preditas são geradas a partir da interpolação entre amostras adjacentes do quadro de referência. Como exemplo, a Figura 3.8 (c) apresenta um casamento com um vetor de movimento que utiliza dois valores fracionários, com as amostras sendo geradas por interpolação. No caso do exemplo, o vetor é (0,5 , 0,75).

No padrão H.264/AVC, a estimação de movimento usando ¼ de pixel é obrigatória. Na prática a estimação é realizada com precisão de ¼ de pixel para os componentes de luminância. Para os elementos de crominância a precisão é, na verdade, de $1/8$ de pixel.

A interpolação para ¼ de pixel é realizada em dois passos. No primeiro passo, é realizada a interpolação dos quadros para ½ pixel usando um filtro FIR de seis *taps*, com pesos (1/32, -5/32, 5/8, 5/8, -5/32, 1/32), da forma como está descrito na Figura 3.9. Na Figura 3.9, as posições inteiras são representadas por letras maiúsculas e as de ½ pixel por letras minúsculas.

Como exemplo, a posição **b** da Figura 3.9 é calculada como está indicado em (4).

$$b = (E - 5 \cdot F + 20 \cdot G + 20 \cdot H - 5 \cdot I + J)/32 \qquad (4)$$

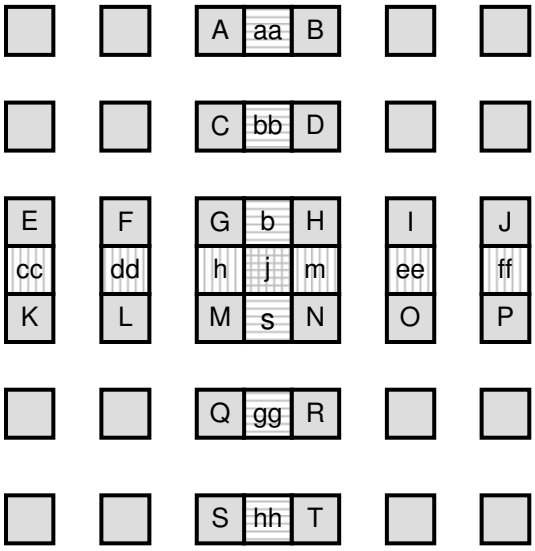

Figura 3.9: Interpolação para posições de ½ pixel para o componente de luminância

De forma similar, **h** é interpolado através da filtragem de **A**, **C**, **G**, **M**, **Q** e **S**. Por outro lado, **j** é gerado através da filtragem de **aa**, **bb**, **b**, **s**, **gg** e **hh**, que foram gerados a partir da primeira filtragem realizada apenas sobre elementos inteiros. É importante ressaltar que o filtro pode ser aplicado horizontalmente ou verticalmente.

No segundo passo, são interpoladas as posições de ¼ de pixel a partir das amostras construídas no primeiro passo e das amostras inteiras, usando a média simples entre dois pontos, na relação que está apresentada na Figura 3.10. Como exemplo, o cálculo da posição **a** na Figura 3.10 é realizado como está apresentado em (5).

$$a = (G + b)/2 \qquad (5)$$

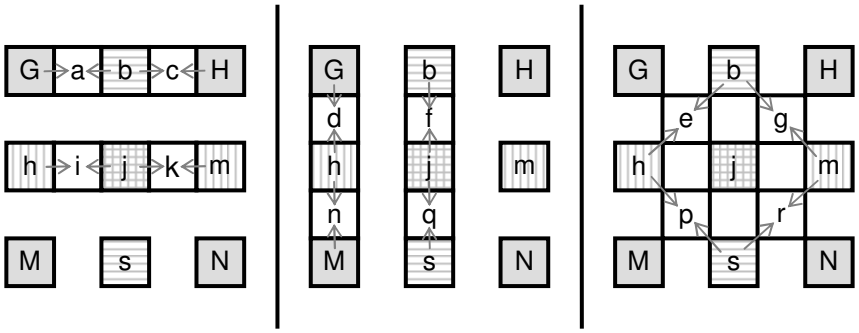

Figura 3.10: Interpolação para posições de ¼ de pixel para o componente de luminância

Para os vetores de crominância, que utilizam uma resolução de $1/8$ de pixel, usa-se interpolação linear. As amostras interpoladas são geradas em intervalos de oito amostras inteiras em cada componente de crominância. Considerando a Figura 3.11 como referência, cada posição interpolada **a** é uma combinação linear das amostras inteiras da vizinhança que, no caso da Figura 3.11, são as posições **A**, **B**, **C** e **D**.

A combinação linear das amostras inteiras é definida pela equação (6).

$$a = round\left\{\left[(8 - d_x)\cdot(8 - d_y)\cdot A + d_x\cdot(8 - d_y)\cdot B + (8 - d_x)\cdot d_y\cdot C + d_x\cdot d_y\cdot D\right]/64\right\} \qquad (6)$$

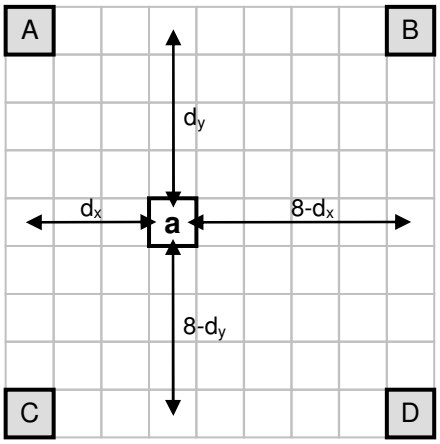

Figura 3.11: Interpolação para componentes de crominância

No exemplo da Figura 3.11, $\mathbf{d_x}$ é igual a três e $\mathbf{d_y}$ é igual a quatro, então **a** é definido por (7) para este exemplo.

$$a = round\left[(20 \cdot A + 12 \cdot B + 20 \cdot C + 12 \cdot D)/64\right] \qquad (7)$$

Outra característica importante do padrão H.264/AVC é o uso de múltiplos quadros de referência. No H.264/AVC os quadros de referência não precisam ser somente os quadros I ou P imediatamente anteriores ou posteriores ao quadro atual. Há a opção de se usar como referência múltiplos quadros para frente ou para trás do quadro atual (RICHARDSON, 2003), como no exemplo apresentado na Figura 3.12.

O padrão H.264/AVC define duas listas contendo os quadros de referência. A lista chamada de lista 0 contém como primeiro quadro o quadro passado mais próximo, seguido de quaisquer outros quadros passados, seguidos de quaisquer outros quadros futuros. A lista chamada lista 1 contém como primeiro quadro o quadro futuro mais próximo, seguido de quaisquer quadros futuros, seguidos de quaisquer outros quadros passados.

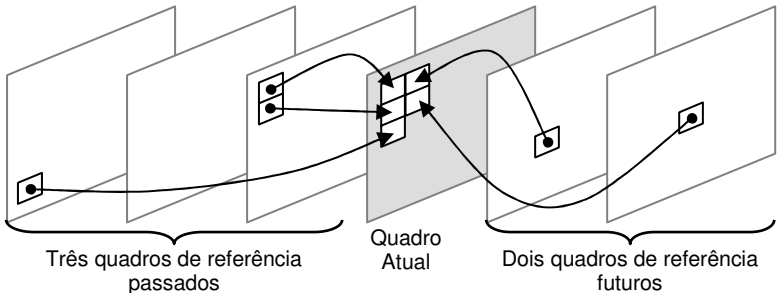

Figura 3.12: Uso de múltiplos quadros de referência.

É importante destacar que o padrão H.264/AVC permite que o codificador escolha uma ordem dos quadros para codificação completamente diferente da ordem dos quadros para apresentação do vídeo (KWON, 2005). Por isso, é possível armazenar nas listas 0 e 1, quadros futuros que, neste caso, são quadros futuros para a apresentação do vídeo, mas são quadros que já foram codificados.

As partições de macroblocos em um *slice* tipo I não utilizam estimação de movimento, uma vez que suas predições são sempre do tipo intra-quadro, isto é, são realizadas apenas sobre amostras internas ao *slice* atual.

As partições de macroblocos em um *slice* tipo P podem ser codificados através de predições intra-quadro, de predições inter-quadros ou através de macroblocos do tipo *skipped*. A predição inter-quadros utiliza a estimação de movimento e pode ser realizada sobre quadros passados ou futuros que tenham sido codificados anteriormente ao quadro atual, armazenados na lista 0. A predição Intra-quadro é a mesma utilizada para *slices* tipo I. Um macrobloco do tipo *skip* pode ser utilizado em *slices* tipo P ou B. Neste caso, nenhuma informação é transmitida para este macrobloco, nem mesmo o resíduo ou o vetor de movimento. Macroblocos do tipo *skip* são gerados quando o custo da taxa de distorção para codificar o macrobloco é mais elevado do que o custo de não enviar informação alguma sobre o macrobloco (KANNANGARA, 2005). Quando o decodificador encontra um macrobloco tipo *skip* em um *slice* P, o decodificador reconstrói o macrobloco a partir do primeiro quadro armazenado na lista 0, calculando seu vetor de movimento a partir dos vetores vizinhos. Se o macrobloco *skip* estiver em um tipo *slice* B, então o macrobloco é reconstruído no decodificador usando predição direta (SAHAFI, 2005).

As partições de macroblocos em um *slice* B são codificadas através de predições inter-quadros que podem ser geradas de diversas formas. É possível realizar a predição utilizando um quadro da lista 0, um quadro da lista 1 ou, ainda, utilizar um quadro de cada lista. Esta última opção é chamada de estimação bi-preditiva. Também é possível realizar a chamada estimação direta. Além destas opções, macroblocos do tipo *skip* também podem ser utilizados, como nos *slices* do tipo P.

A estimação bi-preditiva utiliza um bloco de referência que é construído a partir de dois blocos, um na lista 0 e outro na lista 1. Neste caso, são necessários dois vetores de movimento, um para o quadro de referência da lista 0 e outro para o quadro de referência da lista 1. Então, cada amostra do bloco de predição é calculada como uma média entre as amostras dos quadros das listas 0 e 1. Este cálculo é apresentado em (8), onde **pred0(*i,j*)** e **pred1(*i,j*)** são amostras derivadas dos quadros de referência das listas 0 e 1 respectivamente e onde **pred(*i,j*)** é a amostra bi-preditiva.

$$pred(i,j) = \lfloor (pred0(i,j) + pred1(i,j) + 1)/2 \rfloor \qquad (8)$$

A equação (8) é válida se a estimação bi-preditiva utilizar predição ponderada. A predição ponderada utiliza pesos diferentes sobre as amostras dos macroblocos dos *slices* P ou B. Então, a predição é construída a partir de uma combinação linear destas predições.

Quando a predição direta é usada em um quadro B, nenhum vetor de movimento é transmitido. Ao invés disso, o decodificador calcula os vetores da lista 0 e da lista 1 baseado nos vetores previamente codificados.

A estimação de movimento do padrão H.264/AVC possui ainda características adicionais que contribuem para o aumento da eficiência de codificação. Uma destas características é a possibilidade de utilizar vetores de movimento que apontam para fora dos limites dos quadros. Neste caso é realizada uma extrapolação dos quadros nas bordas. Outra característica interessante é que quadros do tipo B podem ser usados como referência na predição, diferente do que acontece no padrão MPEG-2 (ITU-T, 1994). Por fim, para minimizar o custo da codificação dos vetores de movimento o padrão H.264/AVC prevê que os vetores sejam preditos a partir dos vetores calculados para as partições de macroblocos vizinhas.

Do que foi exposto até agora nesta seção é possível perceber que a estimação de movimento no padrão H.264/AVC é realmente muito complexa, por permitir o uso de técnicas variadas com o objetivo de atingir elevadas taxas de compressão. Uma questão importante e que deve ser considerada no projeto de codificadores H.264/AVC em hardware é que o padrão H.264/AVC normatiza apenas o decodificador. Por isso, a implementação do codificador pode ser realizada com vários graus de liberdade, como já foi citado. Então, várias das características da estimação de movimento previstas pelo padrão podem ser simplesmente ignoradas na construção do codificador para que a complexidade da compressão seja reduzida. Medidas de simplificação deste tipo podem gerar um impacto negativo na eficiência da codificação.

Existem diversos algoritmos utilizados para realizar a busca nos quadros de referência. Estes algoritmos variam desde a busca exaustiva (BHASKARAN, 1999) até algoritmos sub-ótimos que reduzem muito a complexidade da busca, mas que produzem impactos negativos na eficiência da codificação. Dentre estes algoritmos rápidos é possível citar a busca circular (TOURAPIS, 1999), a busca logarítmica (JAIN, 1981), a busca em três passos (LI, 1994), a busca espiral (KHUN, 1999; KROUPIS, 2005), a busca em diamante (YI, 2005) a busca hierárquica (CHU, 1997), entre outros. O apêndice B desta tese descreve alguns destes algoritmos.

Além dos algoritmos de busca é importante destacar a importância dos critérios de similaridade utilizados para decidir qual é o melhor casamento na procura sobre o quadro de referência. Existem vários critérios reportados na literatura dentre os quais cabe citar o erro quadrático médio (MSE), o erro absoluto médio (MAE), a soma de diferenças absolutas (SAD), também chamada de soma de erros absolutos (SAE) (KUHN, 1999), entre outras.

3.6.2 O Módulo da Compensação de Movimento (MC)

A etapa de compensação de movimento (bloco MC nas Figuras 3.4 e 3.5) opera de forma complementar à estimação de movimento. A ME localiza o melhor casamento dentre os quadros de referência e gera um vetor de movimento. É função da compensação de movimento, a partir do vetor de movimento gerado pela ME, copiar os blocos de melhor casamento na memória de quadros anteriormente codificados (quadros passados e/ou futuros) e montar o quadro predito. Este quadro será subtraído do quadro atual para gerar o quadro de resíduos que passará pela transformada.

A compensação de movimento é realizada tanto no codificador quanto no decodificador. A MC no codificador pode ser simplificada, tendo em vista as possíveis simplificações que podem ser realizadas na ME no codificador, conforme foi apresentado na seção anterior. Por outro lado, no decodificador, a compensação de movimento deve ser completa, isto é, deve estar apta a operar sobre todas as funcionalidades do padrão. Assim, a MC no decodificador deve:

- Tratar múltiplos tamanhos de partições de macroblocos;
- Utilizar múltiplos quadros de referência anteriores e posteriores;
- Interpretar corretamente os vetores construídos com base na predição de vetores;
- Tratar vetores que apontam para fora da borda do quadro;
- Reconstruir os macroblocos considerando uma precisão de ¼ de pixel para os vetores de movimento;

- Reconstruir os macroblocos que utilizam as predições bi-preditiva, ponderada e direta para *slices* do tipo B;
- Reconstruir corretamente os macroblocos do tipo *skip* para *slices* tipo P e B.

Como a ME foi abordada na seção anterior, todas as características da MC citadas acima já foram explicadas e, deste modo, não serão explicadas novamente nesta seção.

3.6.3 O Módulo de Predição Intra-quadro

O módulo de predição intra-quadro do padrão H.264/AVC é responsável por realizar a predição nos macroblocos do tipo I. Esta predição é baseada nos valores previamente codificados do *slice* atual dos pixels acima e à esquerda do bloco a ser codificado. A predição intra-quadro para amostras de luminância pode ser utilizada sobre blocos 4x4 ou 16x16. Existem nove diferentes modos de predição intra-quadro para blocos 4x4 e quatro modos para a predição sobre blocos 16x16 (RICHARDSON, 2003). O módulo de predição intra-quadro está presente nos codificadores e nos decodificadores H.264/AVC (Figuras 3.4 e 3.5).

A utilização de um módulo de predição intra-quadro no domínio espacial é uma inovação no padrão H.264/AVC. Em função da predição intra-quadro e considerando também a predição inter-quadros, a transformada é sempre aplicada num sinal erro de predição. Como já foi mencionado na seção 3.6.2, existe um modo adicional de codificação para macroblocos do tipo I, chamado I_PCM. Neste caso, as amostras da imagem são transmitidas diretamente, sem predição, transformada e quantização (RICHARDSON, 2003).

A parte destacada em cinza na Figura 3.13 mostra um bloco 4x4 de luminância a ser codificado pela predição intra-quadro. As amostras acima e a esquerda (letras A a M na Figura 3.13) foram codificadas e reconstruídas antes deste bloco ser processado. Estas amostras serão utilizadas como referências para a predição intra-quadro.

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Figura 3.13: Identificação das amostras para a predição intra-quadro

A Figura 3.14 apresenta os nove tipos diferentes de codificação no modo intra-quadro para blocos 4x4 de luminância. Os modos 0 e 1 fazem uma simples extrapolação (uma cópia) dos pixels das bordas verticais ou horizontais para todas as posições do bloco. O modo DC (2) faz uma média entre as amostras das bordas e copia o resultado para todas as posições do bloco. Os demais modos (3 a 8) fazem uma média ponderada das amostras das bordas, de acordo com a direção da seta na Figura 3.14.

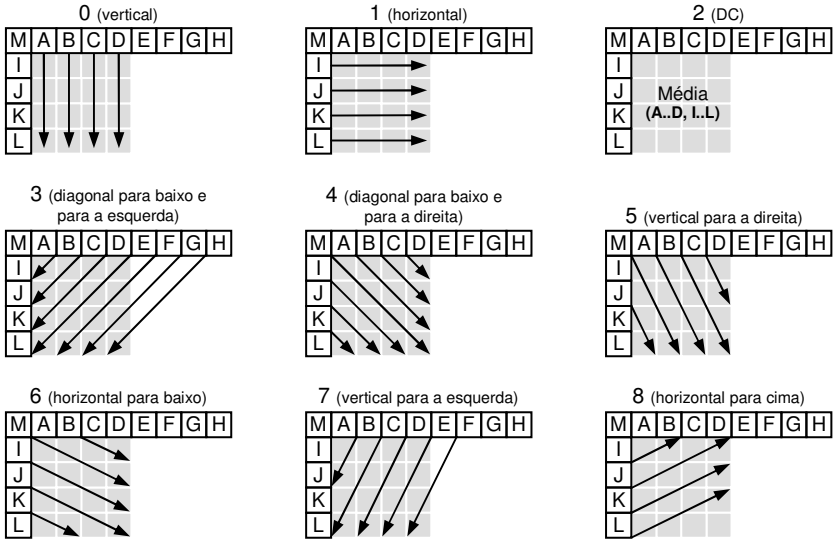

Figura 3.14: Nove modos da predição intra-quadro para blocos de luminância 4x4

Como exemplo, se considerarmos uma predição intra-quadro para blocos 4x4 de luminância no modo 4 da Figura 3.14, então o cálculo da predição da amostra na posição **d** da Figura 3.13 é dado por (9). Por outro lado, o cálculo que gera o valor das posições **a**, **f**, **k** e **p** da Figura 3.13 é dado por (10).

$$d = round\left[(B + 2 \cdot C + D)/4\right] \qquad (9)$$

$$a, f, k, p = round\left[(I + 2M + A)/4\right] \qquad (10)$$

Como já foi mencionado, a predição intra-quadro também pode acontecer sobre o componente de luminância completo de um macrobloco, ou seja, com blocos do tamanho 16x16. Neste caso, são quatro os modos possíveis de predição intra-quadro, conforme está apresentado na Figura 3.15. Os modos 0 e 1 são simples cópias na vertical ou na horizontal das amostras reconstruídas das bordas. O modo 2 é o modo DC e é formado por uma média simples dos elementos das bordas, cujo resultado é copiado para todas as posições do bloco. O modo 3 aplica uma função linear que considera as amostras horizontais e verticais (H e V na Figura 3.15) das bordas.

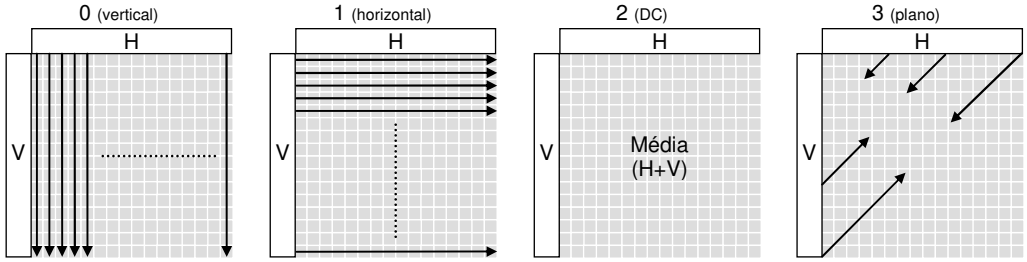

Figura 3.15: Quatro modos da predição intra-quadro para blocos de luminância 16x16

A predição da crominância é realizada diretamente sobre blocos 8x8 e utiliza 4 modos distintos de predição, mas os dois componentes de crominância utilizam sempre o mesmo modo. Os modos de predição para crominância são muito similares aos modos de predição de luminância para blocos 16x16 que foram apresentados na Figura 3.15, exceto pela numeração dos modos. Para a predição de crominância, o modo DC é o zero, o modo horizontal é o um, o modo vertical é o dois e o modo plano é o três (RICHARDSON, 2003).

Os diferentes modos de predição intra-quadro para luminância e crominância possibilitam a geração de uma predição para macroblocos do tipo I que conduz a uma codificação eficiente para este tipo de macrobloco. A escolha de qual modo de predição será utilizado é realizada pelo controle do codificador, que deve sinalizar o modo escolhido no cabeçalho do macrobloco. O módulo do controle será apresentado na seção 3.6.10. Para escolher o melhor modo, o codificador deve gerar a predição sobre todos os modos e escolher qual é o melhor do ponto de vista da eficiência de codificação. Esta tarefa possui uma complexidade computacional elevada para o codificador, porém muito inferior do que a complexidade da estimação de movimento.

3.6.4 O Módulo das Transformadas Diretas (T)

O módulo T, apresentado na Figura 3.4, é responsável pelas transformadas diretas e está presente apenas nos codificadores H.264/AVC. As entradas para o módulo T são blocos 4x4 de resíduos gerados pela etapa de predição. A DCT 2-D direta (FDCT 2-D) é aplicada a todas as amostras de entrada, tanto de luminância quanto de crominância. A FDCT 2-D do padrão H.264/AVC é uma aproximação inteira da DCT 2-D real. Esta aproximação, como já mencionado, foi realizada para reduzir a complexidade do cálculo e evitar erros de casamento entre o codificador e o decodificador. O cálculo da FDCT 2-D inteira é definido no padrão H.264/AVC como está apresentado em (11).

$$Y = C_f X C_f^T \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & & & \\ & X & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 2 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (11)$$

Em (11), **X** é a matriz 4x4 de entrada, $\mathbf{C_f}$ é a matriz da FDCT inteira em uma dimensão, $\mathbf{C_f^T}$ é a transposta de da matriz da DCT e $\mathbf{E_f}$ é a matriz de fatores de escala. O símbolo ⊗ na equação indica uma multiplicação escalar. As letras **a** e **b** na matriz $\mathbf{E_f}$ são as constantes definidas em (12).

$$a = \frac{1}{2}, \qquad b = \sqrt{\frac{2}{5}} \qquad (12)$$

O cálculo da FDCT 2-D transfere para o módulo de quantização (Q), que é o passo que segue a aplicação das transformadas diretas na compressão H.264/AVC, a tarefa de realizar a multiplicação escalar por $\mathbf{E_f}$. Esta tarefa adicional não implica em aumento na complexidade do módulo Q.

Como pode ser observado em (11), as matrizes $\mathbf{C_f}$ e $\mathbf{C_f^T}$ possuem apenas valores -2, -1, 1 e 2. Isso implica na realização de operações bastante simples, consistindo de somas e subtrações diretas e multiplicações por dois, que são equivalentes a um deslocamento de uma casa binária para a esquerda.

O cálculo da FDCT 2-D é aplicado sobre todos os dados de entrada, mas, no caso de amostras com informação de crominância ou com informação de luminância cuja predição tenha sido do tipo intra-quadro 16x16, um cálculo adicional é realizado. Em ambos os casos é aplicada a transformada Hadamard 2-D direta sobre os coeficientes DC resultantes da FDCT 2-D. Esta segunda transformada é uma inovação do padrão

H.264/AVC e foi inserida com o objetivo de atingir ainda mais compressão em áreas homogêneas do vídeo. Os coeficientes DC dos blocos de luminância cuja predição tenha sido intra-quadro 16x16 são submetidos a uma transformada Hadamard 4x4 direta. Por outro lado, uma transformada Hadamard 2x2 direta é aplicada sob os coeficientes DC dos blocos de crominância.

A transformada Hadamard 4x4 direta definida pelo padrão H.264/AVC está apresentada em (13). Esta transformada é aplicada apenas sobre os elementos DC dos blocos 4x4 (resultantes da aplicação da FDCT 2-D) de um macrobloco 16x16 de luminância que tenha utilizado a predição intra-quadro 16x16. Então os 16 elementos DC dos 16 blocos do macrobloco irão formar a matriz 4x4 $\mathbf{W_D}$ de entrada para a Hadamard direta 4x4.

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & W_D & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2 \qquad (13)$$

Como pode ser observado em (13), os elementos das matrizes possuem apenas valores -1 e 1. Deste modo, apenas somas e subtrações são necessárias para realizar os cálculos relativos a esta transformada. A divisão por dois realizada em todos os elementos da matriz de resultados é um simples deslocamento de uma casa binária para a direita.

A transformada Hadamard 2x2 é aplicada apenas para amostras DC de crominância. Esta transformada é utilizada tanto para Cb quanto para Cr. Em função da relação de entrada 4:2:0 para Y, Cb e Cr, cada macrobloco possui 16x16 amostras de luminância, 8x8 amostras de Cb e 8x8 amostras de Cr. As amostras de crominância passam pela FDCT 2-D em blocos de 4x4 amostras. Então, cada matriz 8x8 de entrada é formada por quatro matrizes 4x4. Os quatro elementos DC das quatro matrizes resultantes da FDCT 2-D passam pela Hadamard 2-D 2x2.

O cálculo da Hadamard 2x2 definido pelo padrão H.264/AVC está apresentado em (14).

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} & W_D & \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) \qquad (14)$$

Pode ser observado na fórmula acima que os cálculos da Hadamard 2x2 sobre as amostras DC de Cb e Cr são muito simples, consistindo de poucas somas e subtrações.

O módulo T, sendo formado pelas três transformadas que estão apresentadas em (11), (13) e (14), deve sincronizar a operação entre elas, de modo a gerar o fluxo correto de dados na sua saída. A ordem de processamento está apresentada na Figura 3.16. Inicialmente, o macrobloco de luminância (**Y** na Figura 3.16) passa pela FDCT 2-D. Se o modo é intra-quadro 16x16, então, como foi explicado, os coeficientes DCs das matrizes 4x4 resultantes da FDCT 2-D passam pela Hadamard 4x4 direta. Neste caso, primeiramente é enviado para saída o bloco **–1** na Figura 3.16, com os resultados da Hadamard 4x4 direta, e depois os elementos AC são enviados para a saída (blocos **0** a **15** na Figura 3.16). Se o modo não é intra-quadro 16x16, então os blocos **0** a **15** são enviados diretamente para a saída e a Hadamard 4x4 não é aplicada. Seguindo os 16 blocos de luminância, são processados quatro blocos de crominância Cb e quatro blocos

de crominância Cr. Os blocos de crominância passam pela FDCT 2-D e, sob os elementos DC dos resultados, é aplicada a Hadamard 2-D 2x2. Então é enviado para a saída o bloco **16** na Figura 3.16, com os resultados da Hadamard 2x2 para o componente Cb, depois é enviado o bloco **17**, com os resultados da Hadamard 2x2 para Cr. Finalmente, os coeficientes AC de crominância são enviados para a saída, primeiro os de Cb (blocos **18** a **21**) e depois os de Cr (blocos **22** a **25**).

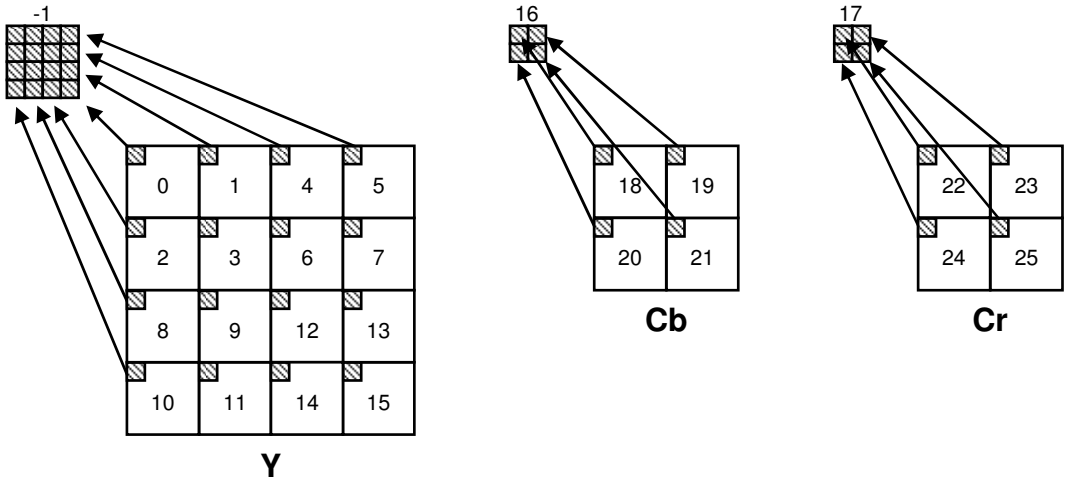

Figura 3.16: Ordem de processamento de amostras pelo módulo T

3.6.5 O Módulo da Quantização Direta (Q)

O módulo Q na Figura 3.4 realiza a quantização direta e a correção da escala do cálculo das transformadas. Este módulo está presente apenas no codificador H.264/AVC. Dependendo do modo de predição utlizado e se o elemento é de crominância ou luminância, os cálculos realizados pelo módulo Q são modificados mas, genericamente, as operações realizadas são uma multiplicação da entrada por uma constante, a soma do resultado com uma outra constante e um deslocamento no resultado da soma controlado por uma terceira constante. Estas constantes são influenciadas diretamente pelo parâmetro de quantização (QP) que é uma entrada externa que informa ao módulo Q qual é o passo de quantização (Qstep) que deve ser utilizado. QP pode variar de 0 a 51 e para cada QP existe um Qstep. Os primeiros seis valores de Qstep, relativos aos seis primeiros QP, são definidos pelo padrão como está apresentado na Tabela 3.4. Os demais Qsteps podem ser derivados dos seis primeiros, pois o Qstep dobra de valor a cada variação de 6 em QP. Então o $Qstep_{(6)}$ é igual $Qstep_{(0)} \times 2$.

Tabela 3.4: Relação entre QP e Qstep

QP	0	1	2	3	4	5	6	...	12
Qstep	0,625	0,6875	0,8125	0,875	1	1,125	1,25	...	2,5

Para os elementos de luminância ou crominância AC e para elementos de luminância DC que não tenham sido codificados no modo intra-quadro 16x16, ou seja, para os elementos que foram processados apenas pela FDCT 2-D no módulo T, a quantização é definida por (15), já considerando o uso de números em ponto fixo:

$$\begin{aligned} |Z_{(i,j)}| &= (|W_{(i,j)}| \cdot MF + f) >> qbits \\ sign(Z_{(i,j)}) &= sign(W_{(i,j)}) \end{aligned} \qquad (15)$$

Em (15), $\mathbf{W_{ij}}$ é o coeficiente resultante da DCT 2-D, **MF** é uma constante gerada a partir do fator de escala e do parâmetro de quantização (**QP**), **f** é uma constante definida pelo padrão em função da predição ter sido gerada pelo modo inter-quadros ou intra-quadro e do parâmetro de quantização utilizado. Por fim, **qbits** indica o deslocamento que deve ocorrer antes do cálculo ser finalizado. É importante salientar que o sinal do resultado deve ser o mesmo sinal da amostra de entrada e que o cálculo é realizado apenas considerando o módulo da amostra de entrada.

A constante **MF** é definida como está apresentado em (16).

$$MF = \frac{PF}{Qstep} << qbits \qquad (16)$$

Em (16), **PF** é o fator de escala, **Qstep** é o passo de quantização e **qbits** é o mesmo deslocamento que estava apresentado em (15). O fator de escala **PF** depende da posição da amostra no módulo e pode ser $\mathbf{a^2}$, **ab/2** ou $\mathbf{b^2/4}$, sendo que **a** e **b** são as mesmas constantes definidas para o módulo T em (12).

O cálculo de **qbits** é função de **QP** e está apresentado em (17). É importante destacar o arredondamento para baixo do fator **QP/6** que está apresentado em (17).

$$qbits = 15 + \lfloor QP/6 \rfloor \qquad (17)$$

Por fim, a constante **f** é definida em (18).

$$\begin{aligned} f &= 2^{qbits}/3 && \text{se a predição for intra} \\ f &= 2^{qbits}/6 && \text{se a predição for inter} \end{aligned} \qquad (18)$$

A quantização para amostras **DC** de crominância ou para amostras de luminância que foram codificados segundo a predição intra-quadro no modo 16x16 é definida por (19).

$$\begin{aligned} |Z_{D(i,j)}| &= (|Y_{D(i,j)}| \cdot MF_{(0,0)} + 2f) >> (qbits + 1) \\ sign(Z_{D(i,j)}) &= sign(Y_{D(i,j)}) \end{aligned} \qquad (19)$$

Este cálculo é bastante similar ao apresentado em (15), sendo que as constantes **MF**, **qbits** e **f** são definidas de modo idêntico ao descrito (16), (17) e (18).

3.6.6 O Módulo das Transformadas Inversas (T^{-1})

O módulo T^{-1} do padrão H.264/AVC, apresentado nas Figuras 3.4 e 3.5, por fazer as operações inversas ao módulo T, possui muitas características idênticas a este módulo. O módulo T^{-1} está presente nos codificadores e nos decodificadores H.264/AVC. O curioso é que o padrão H.264/AVC definiu que a operação do módulo T^{-1} seja fracionada em duas partes. A primeira é realizada diretamente sobre os resultados da quantização direta (Q) e envolve as transformadas Hadamard 2x2 e 4x4 inversas. Este resultado é, então, entregue para a quantização inversa (Q^{-1}). Então, os resultados da quantização inversa são entregues novamente para o módulo T^{-1}, que realiza a última etapa de seus cálculos, aplicando a DCT 2-D inversa (IDCT 2-D).

Para coeficientes DC das informações de crominância ou das informações de luminância cuja predição tenha sido do tipo intra-quadro 16x16 é aplicada a

transformada Hadamard 2-D inversa. Nos coeficientes DC dos blocos de luminância codificados no modo intra-quadro 16x16 é aplicada a Hadamard 4x4 inversa, enquanto que para os coeficientes DC de crominância é aplicada a Hadamard 2x2 inversa.

O cálculo da Hadamard 4x4 inversa está apresentado em (20).

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \quad \\ Z_D \\ \quad \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \qquad (20)$$

Como pode ser observado em (20), o cálculo da Hadamard 4x4 inversa é muito semelhante ao cálculo da Hadamard 4x4 direta que foi apresentada em (13), na descrição do módulo T. As matrizes possuem apenas valores 1, positivos e negativos, como na Hadamard 4x4 direta. Esta característica implica na realização apenas de operações de somas e subtrações. A diferença entre a Hadamard 4x4 direta e inversa está na divisão por dois realizada sob todos os valores de saída, que não existe na Hadamard 4x4 inversa.

A transformada Hadamard 2x2 inversa é aplicada apenas para amostras DC de crominância. Esta transformada é utilizada tanto para Cb quanto para Cr. O cálculo da Hadamard 2x2 inversa é idêntico ao cálculo da Hadamard 2x2 direta, que foi apresentado em (14).

A transformada IDCT 2-D definida pelo padrão H.264/AVC está apresentada em (21), onde **X** é a matriz 4x4 de entrada, $\mathbf{C_i}$ é a matriz da IDCT inteira em uma dimensão, $\mathbf{C_i^T}$ é a transposta de da matriz da IDCT em uma dimensão e $\mathbf{E_i}$ é a matriz de fatores de escala. O símbolo $\otimes$ na equação indica uma multiplicação escalar. As letras **a** e **b** na matriz $\mathbf{E_i}$ são as mesmas constantes definidas para a DCT 2-D direta.

$$Y = C_i^T (X \otimes E_i) C_i = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} \quad \\ X \\ \quad \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \qquad (21)$$

O cálculo da IDCT 2-D já recebe em suas entradas o cálculo relativo à multiplicação escalar por $\mathbf{E_i}$. O módulo de quantização inversa (Q^{-1}), do mesmo modo que para o cálculo da FDCT 2-D, realiza esta operação. Novamente, esta tarefa adicional não implica em aumento na complexidade do módulo Q^{-1}.

Como pode ser observado em (21), as matrizes $\mathbf{C_i^T}$ e $\mathbf{C_i}$ possuem apenas valores 1 e ½, positivos e negativos. Isso implica na realização de operações de somas e subtrações e um deslocamento de uma casa para a direita, no caso do cálculo utilizar o valor ½.

O cálculo da IDCT 2-D é aplicado sobre todos os dados que são entregues pela quantização inversa (Q^{-1}).

O módulo T^{-1}, sendo formado pelas três transformadas inversas que estão apresentadas em (14), (20) e (21) deve sincronizar a operação entre elas, de modo a gerar o fluxo correto de dados na sua saída. A ordem de processamento é exatamente a mesma apresentada na descrição do módulo T.

3.6.7 O Módulo da Quantização Inversa (Q^{-1})

O módulo Q^{-1}, apresentado nas Figuras 3.4 e 3.5, está presente nos codificadores e nos decodificadores H.264/AVC. Este módulo realiza a quantização inversa e a correção da escala do cálculo das transformadas.

Como já mencionado na seção anterior, por uma definição do padrão, antes da quantização inversa são realizados partes dos cálculos relativos ao módulo T^{-1}. As tranformadas Hadamard inversas 4x4 e 2x2, quando utilizadas (DCs de crominância ou DCs de luminância para o modo intra-quadro 16x16), são aplicadas antes da quantização inversa.

Dependendo do modo de predição utilizado e se o elemento é de crominância ou de luminância, os cálculos realizados pelo módulo Q^{-1} são modificados, do mesmo modo que ocorre no módulo Q. Também no módulo Q^{-1} as operações realizadas são uma multiplicação da entrada por uma constante, a soma do resultado com uma outra constante e um deslocamento no resultado da soma que é controlado por uma terceira constante.

Como na quantização direta, estas constantes são influenciadas diretamente pelo parâmetro de quantização (QP) que é uma entrada externa que informa ao módulo Q^{-1} qual é o passo de quantização (Qstep) que foi utilizado na codificação. Os Qsteps utilizados na quantização inversa são os mesmos que são usados na quantização direta.

Para os elementos de luminância ou crominância AC e para elementos de luminância DC que não tenham sido codificados no modo intra-quadro 16x16, a quantização inversa é definida por (22).

$$W'_{(i,j)} = Z_{(i,j)} \cdot V_{(i,j)} \cdot 2^{\lfloor QP/6 \rfloor} \qquad (22)$$

Em (22), $\mathbf{Z_{(i,j)}}$ é o coeficiente quantizado, $\mathbf{V_{(i,j)}}$ é uma constante gerada a partir do fator de escala ($\mathbf{PF_{(i,j)}}$) e do parâmetro de quantização (**QP**) e $2^{\lfloor QP/6 \rfloor}$ é um deslocamento definido por QP e que deve ocorrer na saída antes do cálculo ser finalizado.

A constante $\mathbf{V_{(i,j)}}$ está definida em (23), onde **Qstep** é o passo de quantização e $\mathbf{PF_{(i,j)}}$ é o fator de escala.

$$V_{(i,j)} = (Qstep \cdot PF_{(i,j)} \cdot 64) \qquad (23)$$

O fator de escala $\mathbf{PF_{(i,j)}}$ é diferente na quantização inversa em relação à quantização direta, mas também depende da posição da amostra no bloco. Na quantização inversa $\mathbf{PF_{(i,j)}}$ pode assumir os valores $\mathbf{a^2}$, **ab** ou $\mathbf{b^2}$ sendo que **a** e **b** são as mesmas constantes definidas para o módulo T em (12). A multiplicação por 64, que é um deslocamento de seis casas binárias para a esquerda, é utilizada para prevenir erros de arrendodamento.

A quantização inversa para elementos **DC** de luminância que foram codificados segundo a predição intra-quadro no modo 16x16 é definida em (24).

$$\begin{aligned} W'_{D(i,j)} &= W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{floor(QP/6)} && (QP \geq 12) \\ W'_{D(i,j)} &= \left[W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{1-floor(QP/6)}\right] >> \lfloor QP/6 \rfloor && (QP < 12) \end{aligned} \qquad (24)$$

Este cálculo é bastante similar aos anteriores, sendo que a constante $\mathbf{V_{(i,j)}}$ é definida de modo idêntico ao citado acima.

A quantização para elementos **DC** de crominância é definida por (25).

$$\begin{aligned} W'_{D(i,j)} &= W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{\lfloor QP/6 \rfloor - 1} \qquad (QP \geq 6) \\ W'_{D(i,j)} &= \left[W_{QD(i,j)} \cdot V_{(0,0)} \right] >> 1 \qquad (QP < 6) \end{aligned} \tag{25}$$

3.6.8 O Módulo do Filtro Redutor do Efeito de Bloco

O módulo Filtro nas Figuras 3.4 e 3.5 é o filtro redutor de efeito de bloco normatizado pelo padrão H.264/AVC e que está presente nos codificadores e nos decodificadores que seguem este padrão.

O objetivo deste filtro é suavizar o efeito de blocos do quadro reconstruído antes de ele ser usado para fazer a predição de um novo macrobloco do tipo inter-quadros. Este filtro é adaptativo, distinguindo uma aresta real da imagem, que não deve ser filtrada, de um artefato gerado por um elevado passo de quantização, que deve ser filtrado.

A filtragem é aplicada para bordas verticais e horizontais dos blocos 4x4 de um macrobloco, de acordo com os passos 1 a 4 apresentados abaixo e na Figura 3.17.

1) Filtrar as quatro bordas verticais do componente de luminância (**a**, **b**, **c** e **d** na Figura 3.17);
2) Filtrar as quatro bordas horizontais do componente de luminância (**e**, **f**, **g** e **h** na Figura 3.17);
3) Filtrar as duas bordas verticais de cada componente de crominância (**i** e **j** na Figura 3.17);
4) Filtrar as duas bordas horizontais de cada componente de crominância (**k** e **l** na Figura 3.17).

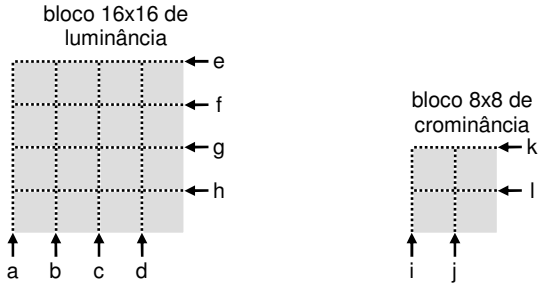

Figura 3.17: Ordem de filtragem de bordas em um macrobloco

Cada operação de filtragem afeta até três amostras de cada lado da borda, como está apresentado na Figura 3.18. Neste caso, são considerados dois blocos 4x4 adjacentes chamados de **p** e **q**, e apenas quatro amostras de cada bloco estão presentes em cada exemplo.

A "força" de filtragem a ser realizada depende da quantização utilizada no bloco, do modo de codificação dos blocos vizinhos e do gradiente das amostras da imagem através da borda. A aplicação do filtro proporciona um aumento significativo da qualidade subjetiva do vídeo reconstruído.

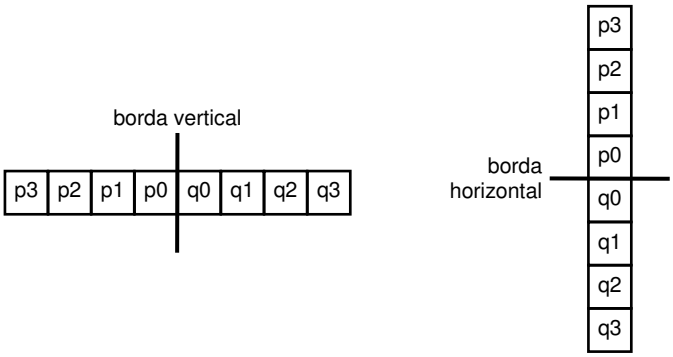

Figura 3.18: Amostras adjacentes para bordas verticais e horizontais

Existem cinco diferentes forças de filtragem, sendo definidos pelo parâmetro ***b*S** (*boundary strength*). O parâmetro ***b*S** pode variar de zero a quatro e, portanto, existem cinco diferentes forças de filtragem, onde um ***b*S**=4 define uma filtragem com força máxima e um ***b*S**=0 define que nenhuma filtragem é realizada. A definição de qual tipo de filtragem deve ser realizada segue as seguintes regras, considerando uma borda entre os blocos **q** e **p**:

- ***b*S=4** : é utilizado se os blocos **q** e/ou **p** foram codificados no modo intra-quadro e se a borda é uma borda de macrobloco.
- ***b*S=3** : é utilizado se os blocos **q** e/ou **p** foram codificados no modo intra-quadro e se a borda não é uma borda de macrobloco.
- ***b*S=2** : é utilizado se os blocos **q** e/ou **p** não foram codificados no modo intra-quadro e se **p** e **q** possuem coeficientes codificados.
- ***b*S=1** : é utilizado se os blocos **q** e/ou **p** não foram codificados no modo intra-quadro; se **p** e **q** não possuem coeficientes codificados; se **p** e **q** usam quadros de referência diferentes ou diferentes números de quadros de referência ou possuem valores de vetores de movimento que se diferenciam por uma ou mais amostras de luminância;
- ***b*S=0** : é utilizado se nenhuma das situações anteriores for verdadeira.

Mas a decisão de realizar a filtragem não depende apenas do parâmetro ***b*S**. Considerando um grupo de amostras (p2, p1, p0, q0, q1, q2), a filtragem acontece se *b*S>0 e se a condição apresentada em (26) for verdadeira.

$$|p0 - q0| < \alpha \quad \text{e} \quad |p1 - p0| < \beta \quad \text{e} \quad |q1 - q0| \leq \beta \qquad (26)$$

Em (26), α e β são definidos pelo padrão e crescem de acordo com a média do parâmetro de quantização (QP) dos blocos **p** e **q**. Se QP possuir um valor baixo, então α e β também terão valores baixos. Neste caso, o efeito de bloco gerado tem importância menor e, por isso, o filtro permanecerá inativo durante mais tempo. Por outro lado, se QP possuir um valor elevado, então a quantização gerará mais perdas, gerando efeitos de blocos mais significativos. Neste caso, os valores de α e β serão maiores, fazendo com que o filtro fique mais tempo ativo e, assim, mais amostras da borda são filtradas.

3.6.9 O Módulo de Codificação de Entropia

Na codificação de entropia, que está presente nos codificadores e decodificadores (Figuras 3.4 e 3.5) o padrão H.264/AVC introduz algumas ferramentas que aumentam bastante a sua eficiência de codificação. Nos níveis hierárquicos superiores (quadros, etc.), os elementos sintáticos são codificados usando códigos binários fixos ou de comprimento variável. A partir do nível de *slices* ou abaixo (macroblocos, blocos, etc.), os elementos sintáticos são codificados usando a codificação aritmética adaptativa ao contexto (CABAC) (RICHARDSON, 2003) ou códigos de comprimento variável (VLC) (SALOMON, 2000). No caso do uso de VLC, a informação residual (coeficientes das transformadas quantizados) é codificada usando a codificação de comprimento variável adaptativa ao contexto (CAVLC) (RICHARDSON, 2003), enquanto que as demais unidades de codificação são codificadas usando códigos Exp-Golomb (SALOMON, 2000). A opção pelo CABAC não está disponível nos perfis *baseline* e *extended*.

A principal inovação introduzida na codificação de entropia do padrão H.264/AVC, tanto na codificação aritmética quanto na codificação VLC é o uso de codificação adaptativa baseada em contextos. Nela, a maneira com que são codificados os diversos elementos sintáticos depende do elemento a ser codificado, da fase em que se encontra o algoritmo de codificação e dos elementos sintáticos que já foram codificados.

Os resíduos resultantes da quantização devem ser reordenados antes de serem utilizados pela codificação de entropia. Os blocos 4x4 com coeficientes do módulo das transformadas, após a quantização, são reordenados em forma de ziguezague, como está apresentado na Figura 3.19. Quando a predição intra-quadro 16x16 é utilizada o bloco -1 da Figura 3.16, contendo os resultados quantizados da Hadamard 4x4 direta, é também ordenado em ziguezague. Os blocos 0 a 15 na Figura 3.16 irão possuir 15 elementos ao invés de 16, em função da aplicação da Hadamard 4x4 sobre os elementos DC de cada bloco. Então os 15 coeficientes AC que restam em cada bloco são reorganizados do mesmo modo que está apresentado na Figura 3.19, mas iniciando da segunda posição. Os blocos 16 e 17 da Figura 3.16, que foram processados pela Hadamard 2x2 direta, são também ordenados em ziguezague, mesmo contendo apenas 2x2 amostras. Finalmente, os 15 coeficientes AC restante nos blocos 4x4 de crominância são, então, reorganizados seguindo a ordem da Figura 3.19, mas iniciando da segunda posição.

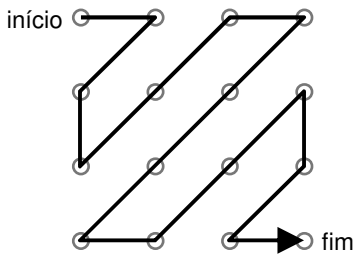

Figura 3.19: Ordem ziguezague de leitura dos blocos 4x4
para a codificação de entropia

As próximas subseções do texto irão apresentar resumidamente, a codificação Exp-Golomb, a codificação de comprimento variável adaptativa ao contexto (CAVLC) e a codificação aritmética binária adaptativa ao contexto (CABAC).

3.6.9.1 Códigos Exp-Golomb

Os códigos Exp-Golomb (*Exponencial* Golomb) (SALOMON, 2000), são códigos de comprimento variável com uma construção regular. Um número inteiro não negativo *N* é codificado usando a estrutura apresentada em (27):

$$Código = [\ M\ zeros\ ][\ 1\ ][\ INFO\ ] \qquad (27)$$

Em (27), **M** representa o número de zeros que antecedem o primeiro valor 1 no código. O valor de **M** é dado por (28)

$$M = \lfloor \log_2 \cdot (num\_cod + 1) \rfloor \qquad (28)$$

A variável **num_cod** em (27) indica qual é o número do código. Códigos com maior probabilidade de ocorrência possuem um número de código mais baixo.

O campo INFO em (28) possui **M** bits e indica qual é a informação codificada. O valor do campo INFO é dado por (29).

$$INFO = num\_cod + 1 - 2^M \qquad (29)$$

O primeiro código de Exp-Golomb não possui os campos **M zeros** e **INFO**, sendo sempre representado apenas pelo valor '1'. A Tabela 3.5 apresenta os seis primeiros códigos de Exp-Golomb.

Tabela 3.5: Seis primeiros códigos de Exp-Golomb

num_cod	Código
0	1
1	010
2	011
3	00100
4	00101
5	00110
. . .	. . .

No padrão H.264/AVC, para cada elemento sintático **k** a ser codificado com o código Exp-Golomb, há uma regra que mapeia o valor de **k** para valores inteiros não negativos, isto é, a regra indica como o valor do elemento sintático **k** deve ser mapeado para um valor de **num_cod**. São quatro as possibilidades de mapeamento, dependendo do tipo de elemento sintático codificado (ITU-T, 2005).

3.6.9.2 Codificação de Comprimento Variável Adaptativa ao Contexto – CAVLC

A codificação de comprimento variável adaptativa ao contexto (CAVLC) é utilizada para codificar os resíduos resultantes da quantização, já ordenados em ziguezague. A CAVLC produz códigos de comprimento variável que são dependentes do contexto da codificação, isto é, da fase em que o algoritmo de codificação se encontra e dos valores que já foram codificados. A CAVLC foi desenvolvida para explorar as características dos blocos quantizados, quais sejam (RICHARDSON, 2003):

- O resultado da quantização produz, tipicamente, matrizes esparsas. Então a CAVLC utiliza RLE (SALOMON, 2000) para representar compactamente as seqüências de zeros.
- Os coeficientes não zeros de alta freqüência, depois da leitura em ziguezague, são frequentemente seqüências de ±1. Então a CAVLC sinaliza o número dos coeficientes ±1 de alta freqüência de uma forma compacta.
- O número de coeficientes não zero em blocos vizinhos são correlacionados. O número de coeficientes é codificado usando uma tabela e a escolha de qual valor da tabela deve ser usado depende do número de coeficientes não zero nos blocos vizinhos.
- O nível (magnitude) dos coeficientes não zero tende a ser maior para os primeiros coeficientes lidos em ziguezague (baixas freqüências) e menores para as freqüências mais elevadas. Então o CAVLC tira vantagem desta característica adaptando a escolha da tabela de VLC que será utilizada para o parâmetro de nível dependendo dos níveis de magnitude dos coeficientes recentemente codificados.

A CAVLC está disponível para todos os perfis do padrão H.264/AVC e é uma alternativa menos complexa ao codificador aritmético adaptativo ao contexto (CABAC), que está disponível nos perfis *main* e *high*. A escolha pelo CAVLC ao invés do CABAC conduz a uma implementação de menor complexidade, mas gera um impacto negativo na eficiência de codificação.

3.6.9.3 Codificação Aritmética Binária Adaptativa ao Contexto (CABAC)

A codificação aritmética binária adaptativa ao contexto (CABAC) é uma ferramenta de codificação disponível apenas para os perfis *main* e *high* do padrão H.264/AVC. A CABAC atinge elevadas taxas de compressão através da seleção dos modelos de probabilidade para cada elemento sintático de acordo com o contexto deste elemento, então as estimativas de probabilidade são adaptadas com base nas estatísticas locais e, finalmente, a codificação aritmética é usada para codificar o elemento sintático.

A codificação pelo CABAC envolve os seguintes estágios:

1. **Binarização**: O CABAC utiliza codificação aritmética binária, o que significa que apenas decisões binárias (0 ou 1) são codificadas. Então os símbolos que não possuem valores binários são binarizados ou convertidos em um código binário antes da codificação. Esse processo é similar ao de converter um símbolo de dados em um código de comprimento variável (VLC), mas o código binário é codificado adicionalmente pelo codificador aritmético antes de ser transmitido. Cada posição de um dígito binário é chamada de um **bin**. Os passos 2, 3, e 4 são repetidos para todos **bins**.
2. **Seleção dos modelos probabilísticos:** Um modelo de contexto é um modelo probabilístico para um ou mais **bins** do símbolo binarizado. A escolha do modelo de contexto a ser utilizado depende dos modelos disponíveis e das estatísticas dos símbolos recentemente codificados. O modelo de contexto armazena a probabilidade de cada **bin** ser 0 ou 1. No H.264/AVC são usados 398 contextos diferentes (ITU-T, 2005; MARPE, 2003).

3. **Codificação aritmética**: Um codificador aritmético codifica cada **bin** de acordo com o modelo probabilístico selecionado. A codificação aritmética será apresentada com mais detalhes no decorrer desta seção, mas é importante notar que, de acordo com os princípios da codificação aritmética, existem somente duas sub-faixas possíveis para cada **bin**, correspondentes a '0' e '1'.
4. **Atualização de probabilidades:** O modelo de contexto selecionado é atualizado com base no valor atual codificado. Por exemplo, se o valor do **bin** atual é '0', a contagem de ocorrências de zeros é incrementada.

Como foi apresentado no item 3 acima, a operação do CABAC utiliza a codificação aritmética para codificar os **bins**. Na codificação aritmética, uma mensagem é representada por um intervalo contido entre '0' e '1'. Considerando que a tabela de probabilidades de ocorrência dos símbolos já tenha sido construída, a codificação aritmética segue os seguintes passos:

1. Definir a faixa inicial de probabilidades contendo todos os símbolos. Para *N* símbolos o intervalo é dividido em *N* subintervalos tal que o intervalo correspondente a um símbolo qualquer possui largura igual à sua probabilidade;
2. Ler o próximo símbolo da entrada;
3. Encontrar a sub-faixa do símbolo atual;
4. Expandir esta sub-faixa para que seja usada como nova faixa, entre '0' e '1';
5. Repetir os passos 2 a 4 até que o valor de um dígito pára de variar (intervalo pequeno o suficiente);
6. Transmitir algum valor que esteja na faixa de valores do último símbolo codificado.

A Figura 3.20 apresenta um exemplo de codificação aritmética, considerando a seqüência de símbolos (0, -1, 0, 2). A tabela de probabilidades para este exemplo está apresentada na Tabela 3.6. Na Figura 3.20, os números maiores sem parênteses indicam a divisão da faixa inicial. Os números grandes entre parênteses indicam qual é o símbolo atual que está sendo codificado. Os números menores indicam a distribuição de probabilidades para cada passo da codificação.

Para o exemplo da Figura 3.20, o intervalo final ficou entre 0,3928 e 0,396 e o valor 0,394 foi o escolhido para ser transmitido e representar o conjunto de valores (0, -1, 0, 2). Com este valor e com o conhecimento dos dados apresentados na Tabela 3.6, o decodificador será capaz de reconstruir os símbolos.

Tabela 3.6: Tabela de probabilidades e sub-faixas para os símbolos do exemplo

Símbolo	Probabilidade	Sub-faixa
-2	0,1	0 – 0,1
-1	0,2	0,1 – 0,3
0	0,4	0,3 – 0,7
1	0,2	0,7 – 0,9
2	0,1	0,9 – 1

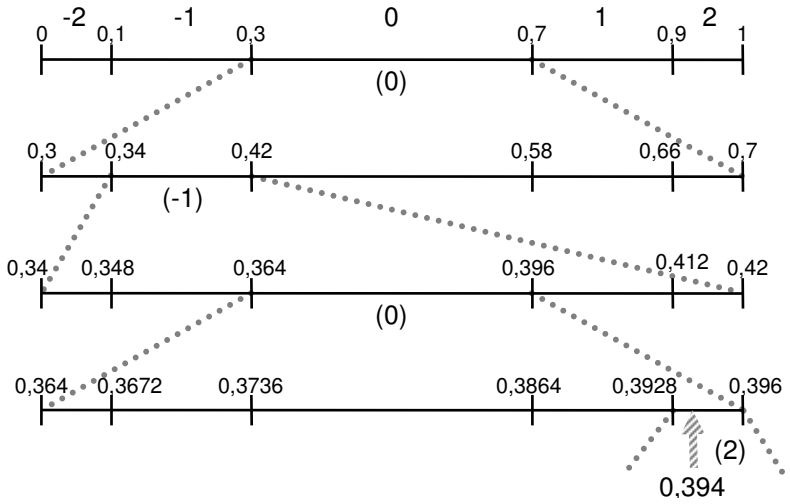

Figura 3.20: Exemplo de codificação aritmética

É importante destacar que o modelo probabilístico usado é independente do codificador aritmético. Isto faz com que possam ser usados modelos probabilísticos diferentes dependendo do estado em que o codificador se encontre. Por exemplo, as probabilidades dos símbolos em cada modelo podem ser atualizadas à medida que os símbolos vão sendo codificados/decodificados.

A codificação/decodificação aritmética requer operações aritméticas para gerar as faixas, o que faz com que a codificação aritmética seja bem mais complexa computacionalmente do que os códigos de comprimento variável. Esta complexidade elevada deve ser tratada tanto pelo codificador quanto pelo decodificador H.264/AVC e é um custo adicional associado ao uso do CABAC. Por outro lado, os ganhos em eficiência de codificação com a utilização do CABAC são significativos. Novamente o compromisso entre complexidade e eficiência de codificação deve ser avaliado para a tomada de decisão de qual tipo de codificador será utilizado na codificação de entropia dos codecs H.264/AVC.

3.6.10 Controle do Codificador

A especificação do padrão H.264/AVC define apenas a sintaxe do *bitstream* e o processo de decodificação. O processo de codificação é deixado de fora do escopo do padrão para permitir maior flexibilidade para as implementações. Mas o controle do codificador é um problema chave para a compressão, pois ele determina quais as decisões de codificação serão tomadas para cada vídeo processado (WIEGAND, 2003).

No codificador H.264/AVC existem muitas decisões que devem ser tomadas pelo codificador, incluindo a definição do tipo de predição (intra-quadro ou inter-quadros), a definição do tamanho de bloco usado nas predições intra-quadro ou inter-quadros, a definição se o modo skip será usado ou não, a definição se será usada bi-predição, etc. Como são muitas as possibilidades, é muito importante que as decisões tomadas sejam as melhores possíveis, pois com escolhas sub-ótimas, alguns dos benefícios trazidos pelo H.264/AVC podem ser perdidos (PURI, 2004).

O problema do controle do codificador é causado porque seqüências de vídeo típicas contêm grande variação de conteúdo e movimento, sendo necessário selecionar entre diferentes opções de codificação com variação na eficiência da relação taxa-distorção para diferentes partes do vídeo. A tarefa do controle do codificador é determinar um

grupo de parâmetros de codificação de modo que um certo compromisso na relação taxa-distorção seja atingido (WIEGAND, 2003).

Mesmo que não sejam normatizadas pelo padrão H.264/AVC, técnicas de otimização taxa-distorção (WIEGAND, 2003; SULLIVAN, 1998) devem ser usadas para todas as decisões tomadas pelo codificador. Com este tipo de controle do codificador, significativos ganhos em termos de eficiência de codificação podem ser obtidos. Mas o uso deste tipo de critério não evita que todas as possíveis combinações de modos de codificação tenham que ser calculadas para que se possa atingir a escolha ótima. Entretanto, ao se fazer a escolha que minimiza o custo **J** de cada decisão, esta escolha vai tender a maximizar a eficiência do codificador.

O parâmetro λ é usado para controlar a taxa obtida e está relacionado com o passo de quantização **Qstep**. Para cada decisão, é medido o custo **J**, de acordo com (30).

$$J = D + \lambda \cdot R \qquad (30)$$

Em (30), **R** é a taxa de bits e **D** é a distorção. Neste caso, para λ fixo, é possível gerar vários custos **J**. Assim as opções podem ser varridas na busca do custo **J** que esteja mais próximo da taxa **R** ou da distorção **D** desejados.

3.7 Análise de Complexidade

A complexidade relativa dos diversos módulos do padrão H.264/AVC foi avaliada por alguns trabalhos publicados na literatura. Neste caso, a complexidade foi avaliada em termos do tempo de processamento exigido em cada módulo, o que equivale ao número aproximado de operações necessárias em cada módulo (HOROWITZ, 2003). Em função da própria estrutura do decodificador padronizado e das possibilidades de implementação previstas para o codificador, é possível, mesmo sem uma avaliação mais criteriosa, identificar que o codificador completo possui uma complexidade sensivelmente superior à complexidade do decodificador. Além disso, analisando o padrão e, mesmo, o texto deste capítulo da tese, é possível estimar, empiricamente, que a operação de maior complexidade do codificador é a estimação de movimento. Por outro lado, no decodificador é possível estimar que o módulo mais complexo seja o da compensação de movimento.

A complexidade computacional de um codec H.264/AVC é significativamente superior aos demais padrões de compressão de vídeo, uma vez que o crescimento na complexidade é o preço pago para atingir as maiores taxas de compressão dentre os padrões existentes.

Alguns trabalhos encontrados na literatura mediram a complexidade relativa dos módulos do H.264/AVC, tomando por base o código de referência disponibilizado pelo JVT (JVT, 2007). Estas análises confirmaram as previsões empíricas no que diz respeito ao nível de complexidade dos módulos do compressor e do descompressor.

O trabalho de Zhang (2003) apresenta uma análise de complexidade dos módulos principais do padrão H.264/AVC. A análise foi realizada sobre o perfil *baseline* considerando tanto o codificador quanto o decodificador. Esta análise utilizou o código de referência rodando sobre um processador de propósito geral. A Tabela 3.7 apresenta um resumo dos resultados percentuais encontrados.

O autor não apresenta os dados de complexidade do módulo de predição intra-quadro para o decodificador (ZHANG, 2003), como pode ser percebido na Tabela 3.7.

De qualquer modo, a estimação/compensação de movimento possui mais de 80% da complexidade total do codificador, o que é impressionante. No decodificador, a compensação de movimento é responsável por quase 45% da complexidade total.

Tabela 3.7: Análise da complexidade dos módulos de um codec H.264/AVC implementado em software

Codificador		Decodificador	
Módulo	Complexidade (%)	Módulo	Complexidade (%)
ME / MC	81,78	MC	44,92
$T / Q / Q^{-1} / T^{-1}$	5,49	T^{-1}/Q^{-1}	22,22
Predição Intra-Quadro	5,29	Filtro	14,8
Filtro	0,82	Outros	18,06
Outros	6,62	-	-

Fonte: ZHANG, 2003.

No trabalho de Huang (2005) a análise de complexidade também foi realizada sobre o perfil *baseline* do H.264/AVC e também utilizou o código de referência rodando sobre um processador de propósito geral. Neste trabalho, apenas a complexidade do codificador foi avaliada e os resultados estão na Tabela 3.8.

Tabela 3.8: Análise da complexidade dos módulos de um codificador H.264/AVC implementado em software

Módulo	Complexidade (%)
ME / MC	57
Predição Intra-Quadro	20
$T / Q / Q^{-1} / T^{-1}$	16
Codificação de Entropia	4
Outros	3

Fonte: HUANG, 2005.

Segundo os resultados apresentados na Tabela 3.8, a estimação/compensação de movimento é responsável por quase 60% da complexidade computacional do compressor H.264/AVC para as simulações realizadas.

A Tabela 3.9 apresenta os resultados da análise de complexidade desenvolvida por Horowitz (2003) para um decodificador H.264/AVC. Os testes realizados para gerar esta tabela também utilizaram o perfil *baseline*. A análise utilizou o código de referência rodando sobre um processador de propósito geral.

A análise apresentada na Tabela 3.9 indica um resultado surpreendente para os testes realizados neste artigo (HOROWITZ, 2003): o filtro redutor de blocagem apresentou a complexidade mais elevada dentre todos os módulos do decodificador. A segunda maior complexidade foi encontrada no módulo da compensação de movimento.

Tabela 3.9: Análise de complexidade dos módulos de um decodificador H.264/AVC implementado em software

Módulo	Complexidade (%)
Filtro	33
MC	25
Codificação de Entropia	13
T^{-1} / Q^{-1}	13
Outros	16

Fonte: HOROWITZ, 2003.

A Tabela 3.10 apresenta a análise desenvolvida por Wu (2005) para um decodificador H.264/AVC. Os dados utilizados para gerar esta tabela foram extraídos do artigo, que apresentava a análise em MIPS. Os testes realizados utilizaram, mais uma vez, o perfil *baseline* e utilizaram o código de referência rodando sobre um processador de propósito geral.

Tabela 3.10: Análise de complexidade dos módulos de um decodificador H.264/AVC implementado em software

Módulo	Complexidade (%)
Filtro	51,32
MC	28,13
Codificação de Entropia	7,25
T^{-1} / Q^{-1}	6,71
Predição Intra-Quadro	3,3
Outros	3,3

Fonte: WU, 2005.

A análise apresentada na Tabela 3.10 indica, mais uma vez, que o filtro redutor de blocagem apresentou a complexidade mais elevada dentre os módulos investigados. O filtro, sozinho, é responsável por mais da metade da complexidade de todo o decodificador. A segunda maior complexidade é da compensação de movimento.

Os dados apresentados nas Tabelas 3.7, 3.8, 3.9 e 3.10 são ligeiramente diferentes entre si. O código de referência apresenta várias opções parametrizáveis que não foram explicitadas pelos autores, de modo que não é possível conhecer quais foram os parâmetros utilizados nas simulações realizadas nestes trabalhos. Além disso, as taxas de complexidade variam de vídeo para vídeo e também variam de acordo com o tamanho de quadro utilizado. Mas ainda mais importante para a definição da complexidade é o perfil do padrão usado na sua avaliação. Por exemplo, a complexidade da compensação de movimento no decodificador cresce significativamente quando é usado o perfil *main* ao invés do perfil *baseline*, em função da bi-predição, do uso do CABAC e de outras ferramentas não disponíveis no perfil *baseline*. Por outro lado, a complexidade do módulo do filtro, por exemplo, permanece inalterada se o perfil *main* for usado ao invés do perfil *baseline*. De qualquer modo, o mais importante é que os resultados das Tabelas 3.7 e 3.8 convergem na constatação de

que os módulos mais complexos no codificador são os da estimação/compensação de movimento, que em ambos os casos, são responsáveis por mais da metade da complexidade total do codificador. As análises destas duas tabelas também convergem ao colocar em segundo lugar na complexidade do codificador o módulo de predição intra-quadro, que sozinho possui uma complexidade equivalente à complexidade conjunta dos módulos das transformadas direta e inversa e das quantizações direta e inversa. Do ponto de vista do decodificador, as Tabelas 3.7, 3.9 e 3.10 apresentam resultados mais discrepantes, principalmente em função da complexidade do filtro redutor do efeito de blocagem, apresentada na Tabela 3.7. Os resultados das Tabelas 3.9 e 3.10 são mais convergentes, mas também apresentam diferenças. Ainda assim, as tabelas indicam que o filtro redutor de blocagem e a compensação de movimento possuem uma contribuição significativa para a complexidade total do descompressor e, juntos, para todos os casos, estes módulos são responsáveis por mais de 50% da complexidade do decodificador.

Comparando os resultados das complexidades dos codificadores e decodificadores H.264/AVC apresentados nas Tabelas 3.7, 3.8, 3.9 e 3.10 é possível perceber que o codificador H.264/AVC é muito mais complexo do que o decodificador. O módulo do filtro redutor de blocagem é idêntico para o codificador e para o decodificador e pode ser usado como parâmetro de comparação. Na tabela 3.7, por exemplo, o filtro contribui com menos de 1% da complexidade do codificador e o mesmo filtro passa a contribuir com quase 15% da complexidade do decodificador, indicando claramente que as operações do decodificador são de uma complexidade significativamente menor que as operações do codificador. A mesma comparação pode ser realizada entre as complexidades do filtro na Tabela 3.8 (<3% no codificador), com o mesmo filtro na Tabela 3.9 (33% no decodificador) e na Tabela 3.10 (51,32% no decodificador).

Existem outras análises de complexidade do padrão publicadas na literatura que consideram o tempo gasto no processamento de cada função do código de referência ou que consideram o número de vezes que cada função é executada como métrica de complexidade como o trabalho realizado por Horowitz (2003), mas estes trabalhos não serão discutidos neste texto.

3.8 Principais Desafios

O padrão H.264/AVC é muito extenso e complexo, como foi possível perceber no decorrer deste texto. Deste modo, implementar um codificador ou decodificador H.264/AVC não é uma tarefa trivial, mesmo que esta implementação seja realizada em software. O problema passa a ser muito mais crítico se a implementação tiver por objetivo atingir tempo real para resoluções mais elevadas, como SDTV ou HDTV.

Em função do que foi exposto na seção 3.7, é possível identificar que o módulo da estimação/compensação de movimento é o gargalo de desempenho do codificador. Por outro lado, os módulos do filtro redutor de blocagem e da compensação de movimento são os gargalos de desempenho do decodificador. Mas todas as análises apresentadas na seção anterior foram desenvolvidas a partir da execução do código de referência sobre um processador de propósito geral. Por isso, esta análise de complexidade não considerou as possíveis adaptações algorítmicas visando aumentar o paralelismo nos módulos do codificador e do decodificador. Este tipo de análise, buscando identificar as dependências de dados dos algoritmos utilizados e, por conseqüência, definindo qual o grau de paralelismo que estes algoritmos permitem, é muito importante para avaliar a

complexidade quando estes algoritmos estão sendo executados em um ambiente multiprocessado ou quando os algoritmos serão implementados em hardware dedicado, com a conseqüente exploração de paralelismo.

A estimação de movimento, que é o módulo de maior complexidade computacional do codificador, quando baseada no algoritmo de busca exaustiva (*full search*), é um exemplo de módulo que pode ser implementado de uma maneira massivamente paralela, pois as dependências de dados são pequenas. Deste modo, tratando o problema da estimação de movimento com múltiplas unidades trabalhando em paralelo, é possível aumentar a taxa de processamento deste módulo e fazer com que os requisitos da aplicação sejam atingidos. É interessante notar que praticamente todos os trabalhos da literatura que visam minimizar a complexidade da estimação de movimento, consideram apenas a redução da complexidade quando um código fonte tipicamente serial é executado em um processador convencional. Algumas destas soluções aumentam a dependência de dados com o objetivo de reduzir o número de instruções executadas sequencialmente. O efeito colateral desta solução é que a exploração do paralelismo fica mais restrita. Assim, é possível que um algoritmo que reduz a complexidade da estimação de movimento (considerando um processador de propósito geral) acabe por atingir uma taxa de processamento inferior aos algoritmos não otimizados, quando ambos forem implementados em hardware. Isso pode ocorrer porque a exploração do paralelismo será dificultada com o algoritmo otimizado se a otimização implicar em um aumento na dependência de dados.

Ainda assim, os resultados da seção anterior espelham os maiores desafios para os projetistas de codecs H.264/AVC em software ou em hardware, quais sejam: encontrar uma solução para a questão da complexidade dos módulos da estimação de movimento, da compensação de movimento, da predição intra-quadro, do filtro de blocagem, etc.

O foco dos projetistas de hardware reside na exploração do paralelismo com o objetivo de atingir tempo real para qualquer resolução de vídeo. Em função da complexidade e extensão do padrão, não existe nenhum módulo simples de ser implementado em hardware para atingir os requisitos da aplicação alvo, pois em cada módulo do codificador ou do decodificador residem desafios de difícil solução.

O projeto de um codificador ou decodificador de vídeo em hardware envolve uma série de compromissos relacionando requisitos, por vezes, contraditórios. Alguns destes compromissos são comuns para os projetistas de software e hardware, mas outros são específicos para os projetistas de hardware. O projeto de um codec de vídeo em hardware deve estar focado, entre outros, nos seguintes requisitos:

- **Taxa de compressão**: o codificador deve ser capaz de gerar um *bitstream* codificado eficientemente, respeitando as exigências de armazenamento e/ou transmissão e/ou display da aplicação.
- **Qualidade do vídeo**: o codificador, mesmo gerando perdas de informação no processo de codificação, deve ser capaz de gerar um *stream* de bits que, ao ser decodificado, apresente uma qualidade objetiva e subjetiva de imagem que esteja de acordo com as exigências da aplicação alvo.
- **Largura de banda para transmissão**: o codificador deve ser capaz de utilizar apenas a largura de banda exigida pela aplicação quando o vídeo comprimido for transmitido. Este requisito está diretamente relacionado com a taxa de compressão e com a qualidade do vídeo.

- **Taxa de processamento**: o codificador ou decodificador deve ser capaz de processar o vídeo de entrada na taxa de amostras por segundo exigida pela aplicação. A exploração do paralelismo através de implementações em hardware contribui para o atendimento deste requisito, mas amplia a utilização de recursos de hardware.
- **Quantidade de recursos de hardware utilizados**: a implementação do codificador ou decodificador deve respeitar as restrições de utilização de recursos de hardware impostas pela aplicação ou pelo custo de fabricação. Deste modo, o projeto deve, respeitando as exigências da taxa de processamento, utilizar a menor quantidade de hardware possível.
- **Largura de banda de memória**: o projeto do codificador e do decodificador deve considerar que um dos maiores gargalos deste tipo de sistema está na comunicação com a memória. Assim, o projeto deve buscar a minimização dos acessos à memória, para respeitar as restrições de largura de banda disponível para a comunicação com a memória. Em projetos completamente dedicados, memórias específicas e até uma hierarquia de memória podem ser implementadas para viabilizar uma comunicação mais efetiva com a memória. Neste caso, a quantidade de recursos de hardware utilizada irá crescer.
- **Energia consumida**: muitas aplicações que utilizam codecs de vídeo são sistemas embarcados, onde a minimização na energia consumida é um requisito fundamental. Neste caso, o projeto do codec pode aplicar algumas técnicas para minimizar o consumo de energia. Contribuem para atender este requisito a minimização dos acessos à memória, a redução na utilização de recursos de hardware, entre outros.

O projeto de um codec de vídeo deve estar focado nestes requisitos e, dependendo da aplicação alvo, deve ser tomada a decisão de quais requisitos serão priorizados, em detrimento dos demais.

No projeto de um codec H.264/AVC esta relação contraditória de requisitos é ainda mais problemática, pois este é o padrão mais complexo desenvolvido até então. Atingir desempenhos satisfatórios (qualidade de vídeo, taxa de compressão, etc.) para resoluções elevadas, como HDTV, por exemplo, é mesmo uma tarefa árdua e, por vezes, inatingível, dependendo das restrições da aplicação alvo.

Em função do estudo do padrão que conduziu a este texto e em função dos trabalhos relacionados que foram apresentados, é possível concluir que existem diversos desafios na implementação de um codec H.264/AVC. Os desafios residentes na implementação dos principais módulos de um codec H.264/AVC estão listados abaixo:

- **Estimação de movimento (codificador)**: este é o módulo que possui a maior complexidade computacional no codificador, exigindo o maior número de operações aritméticas para concluir seus cálculos, além de exigir o maior número de acessos à memória. Existem vários algoritmos sub-ótimos que podem ser aplicados visando reduzir a complexidade e o número de acessos à memória realizados por este módulo, mas o uso destes algoritmos conduz a uma redução na eficiência da codificação. Para alguns destes algoritmos a dependência de dados tende a ser reduzida, permitindo uma elevada paralelização das operações, elevando a taxa de processamento. Outra solução é, deliberadamente, optar por

não implementar todos os diferentes modos de predição permitidos, uma vez que o padrão permite este tipo de decisão. Neste caso, a taxa ou a qualidade de compressão irão cair.

- **Predição Intra-quadro (codificador)**: este módulo possui uma complexidade muito inferior à complexidade da estimação de movimento no codificador, mas ainda assim, por possibilitar até treze diferentes tipos de codificação, este módulo possui uma complexidade capaz de tornar o seu projeto em hardware uma tarefa desafiadora. Um dos maiores problemas é que as amostras utilizadas para realizar a predição intra-quadro são as amostras reconstruídas para o quadro atual. Isso significa que os blocos preditos devem ser processados pelos módulos T, Q, Q^{-1} e T^{-1} antes de serem utilizados como referência para processar os próximos blocos do quadro atual. Isso implica que a predição intra-quadro deve ficar "esperando" para que os dados sejam reconstruídos. Uma possível solução para este problema é desenvolver soluções completamente paralelas para os módulos T, Q, Q^{-1} e T^{-1} , reduzindo o tempo de espera. Outra solução, como foi sugerido na estimação de movimento, é não implementar os treze modos de compressão previstos para a predição intra-quadro. Novamente, a qualidade ou a taxa de compressão irão diminuir.
- **Controle (codificador)**: o controle do codificador é responsável por tomar as decisões sobre a codificação. Para tanto, todos os modos possíveis de codificação devem ser testados e seus custos devem ser avaliados em função da relação entre a taxa-distorção. As decisões em si não são tarefas complexas, mas elas são tomadas sobre dados que precisam ser gerados para todas as possibilidades de codificação disponibilizadas.
- **Transformadas e quantização (codificador):** estes módulos (T, T^{-1}, Q e Q^{-1}), mesmo tendo uma complexidade computacional reduzida se comparado aos outros módulos de um codec H.264/AVC, estão no caminho crítico da predição intra-quadro, como já foi explicado. Deste modo, quanto maior for o desempenho destes módulos menor será o tempo em que a predição intra-quadro deverá ficar parada. Além disso, os módulos das transformadas e quantização diretas e inversas são muito importantes para o controle do codificador, pois as decisões relativas ao melhor tamanho de bloco utilizado são realizadas a partir de comparações sobre a imagem reconstruída. A reconstrução da imagem passa pelos quatro módulos das transformadas e quantização e, deste modo, estão no caminho crítico deste módulo e do codificador como um todo.
- **CABAC (codificador e decodificador)**: o CABAC, presente no codificador e no decodificador não possui uma complexidade computacional muito elevada. O maior problema é que ele opera sobre símbolos binários, isto é, sobre '0' e '1'. Deste modo, para ser processada uma amostra representada em oito bits, por exemplo, são necessárias oito saídas temporalmente distintas do CABAC. Como o CABAC é adaptativo ao contexto, a exploração do paralelismo fica dificultada, uma vez que a codificação do dado futuro depende do dado atual.
- **Filtro redutor de blocagem (codificador e decodificador)**: O filtro redutor de blocagem, como o CABAC, é adaptativo ao contexto, por isso, possui algumas dependências de dados que dificultam a exploração máxima do paralelismo. Além disso, o filtro precisa acessar a memória para gravar os blocos reconstruídos, gerando uma dificuldade a mais na sua implementação, pois o

filtro deverá disputar o acesso à memória com o estimador/compensador de movimento e com a predição intra-quadro.

- **Compensação de movimento (decodificador)**: A compensação de movimento é um dos módulos mais complexos do decodificador, envolvendo muitas operações aritméticas e muitos acessos à memória. O maior desafio no projeto deste módulo reside no fato de que ele deve estar apto a tratar qualquer tipo de informação gerada pela estimação de movimento do codificador. Novamente, a dependência de dados é pequena e, então, o paralelismo pode ser explorado.
- **Predição intra-quadro (decodificador)**: A predição intra-quadro no decodificador não apresenta os mesmos desafios da predição intra-quadro do codificador, uma vez que a dependência de dados relativa à reconstrução do bloco atual existe, mas não é dependente das operações T e Q, que não existem no decodificador. Ainda assim, este módulo é um dos módulos mais complexos do decodificador e demanda um projeto dedicado com algum grau de paralelismo, especialmente para aplicações com resoluções elevadas.

4 ARQUITETURAS DESENVOLVIDAS PARA AS TRANSFORMADAS E QUANTIZAÇÃO

"Nunca pensei que minha sina
Fosse andar longe do pago"

João da Cunha Vargas

Os módulos do padrão H.264/AVC que foram escolhidos para as primeiras investigações neste trabalho foram os módulos das transformadas direta e inversa (T e T^{-1}) e os módulos da quantização direta e inversa (Q e Q^{-1}). Conforme foi descrito no capítulo 3, os módulos T^{-1} e Q^{-1} estão presentes no decodificador e os módulos T, T^{-1}, Q e Q^{-1} estão presentes no codificador.

Este capítulo da tese apresenta um relato resumido das investigações desenvolvidas sobre estes módulos, que envolveram o estudo do estado da arte, a proposição de arquiteturas para construir estes módulos em hardware, a síntese destas arquiteturas para FPGAs Xilinx a partir da sua descrição em VHDL e a análise dos resultados obtidos, incluindo comparações com trabalhos publicados na literatura. O Apêndice A apresenta a metodologia de validação e prototipação das arquiteturas desenvolvidas.

As arquiteturas desenvolvidas buscaram, inicialmente, atender as exigências do padrão H.264/AVC e atingir um elevado desempenho, suficiente para processar televisão digital de alta definição (HDTV) em tempo real. Para tanto, alguns ajustes nos algoritmos propostos foram realizados, para permitir uma maior exploração do paralelismo, para permitir a sincronização destes módulos com os demais módulos do codec H.264/AVC e/ou para permitir facilidades de integração com os outros módulos do H.264/AVC.

As investigações apresentadas neste capítulo incluem o desenvolvimento de diversas arquiteturas para estes módulos. A primeira investigação apresentada é a exploração do espaço de projeto das transformadas, que teve o objetivo de identificar quais seriam as soluções mais interessantes para usar no projeto dos módulos T e T^{-1}. A seguir são apresentadas duas arquiteturas para o módulo das transformadas diretas, uma com taxa de consumo e produção de uma amostra por ciclo de *clock* e outra com taxa de consumo e produção de dezesseis amostras por ciclo de *clock*. Então é apresentada a solução arquitetural para o módulo das transformadas inversas, que possui uma com taxa de consumo e produção de uma amostra por ciclo de *clock*. A última arquitetura apresentada para as transformadas é uma solução capaz de realizar os cálculos de todas as cinco diferentes transformadas do padrão H.264/AVC e com seleção de nível de paralelismo variando de uma até 16 amostras consumidas e produzidas por ciclo de

clock. Finalmente, são apresentadas as soluções arquiteturais para os módulos da quantização direta e inversa, que consomem e produzem uma amostra por ciclo de *clock*.

4.1 Exploração do Espaço de Projeto das Transformadas

As arquiteturas das transformadas FDCT 2-D, IDCT 2-D, Hadamard 2-D 4x4 direta e Hadamard 2-D 4x4 inversa são os componentes mais críticos dos módulos das transformadas (T e T^{-1}) do compressor H.264/AVC.

Esta seção apresenta uma investigação sobre as alternativas arquiteturais para o projeto destas transformadas. Então, várias alternativas de implementações para estas arquiteturas foram desenvolvidas e implementadas, com o intuito de investigar quais eram as melhores arquiteturas para cada codificador, de acordo com a resolução de imagem que este codificador deve tratar, de acordo com a quantidade de recursos de hardware disponíveis, entre outros. Estas avaliações foram realizadas inspiradas em implementações encontradas na literatura (WIEN, 2003; WANG, 2003; LIN, 2005; CHEN, 2005) e em algumas soluções inéditas, desenvolvidas no escopo deste trabalho.

Os experimentos iniciais consideraram apenas a arquitetura da Hadamard 2-D 4x4 direta nesta exploração, pois verificou-se que todas as arquiteturas que processam matrizes de entrada de 4x4 elementos (FDCT 2-D, IDCT 2-D, Hadamard 2-D direta e inversa) possuem uma organização similar e, deste modo, podem ser descritas com uma estrutura muito parecida. Deste modo, a aplicação das alternativas investigadas com a Hadamard 2-D direta podem facilmente ser aplicadas para as demais transformadas que processam matrizes de entrada de 4x4 elementos. Além disso, os experimentos que foram realizados visam, essencialmente, observar as possibilidades distintas das diversas arquiteturas em termos de uso de recursos, de freqüência de operação e de taxa de processamento. Esta observação sobre a implementação da Hadamard 2-D direta se demonstrou satisfatória. Para todas as arquiteturas, foram usados 14 bits para representar a entrada e 17 bits para representar a saída. As diferentes soluções arquiteturais foram descritas em VHDL e mapeadas para FPGAs da família Virtex-II Pro (XILINX, 2005) da Xilinx, através da ferramenta Synplify Pro da Synplicity (SYNPLICITY, 2007).

4.1.1 Arquitetura Serial com Separabilidade

A primeira alternativa explorada para a Hadamard 4x4 direta considera entradas seriais (isto é, uma amostra é processada a cada ciclo de *clock*) e usa a propriedade da separabilidade da transformada, calculando duas transformadas em uma dimensão para obter o resultado da transformada em duas dimensões, onde o resultado da primeira é transposto e a segunda transformada é, então, aplicada. Esta arquitetura realiza exatamente o cálculo que foi apresentado na fórmula (13). A Figura 4.1 apresenta esta implementação.

As transformadas Hadamard em uma dimensão foram projetadas para consumir uma amostra de entrada a cada ciclo de *clock* e os dados da matriz 4x4 de entrada são entregues para a primeira Hadamard 1-D linha a linha. A latência da arquitetura completa de cálculo da Hadamard 2-D 4x4 com separabilidade é de 24 ciclos de *clock*.

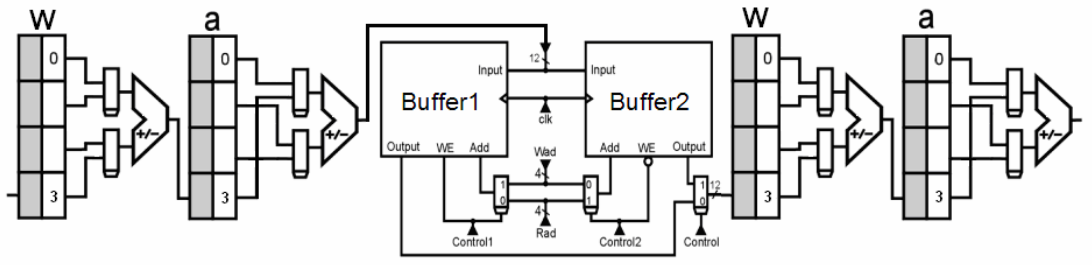

Figura 4.1: Arquitetura da transformada serial com separabilidade

A implementação das transformadas Hadamard 1-D foi projetada para operar em um *pipeline* de dois estágios, sendo que cada estágio consome quatro ciclos de *clock* para finalizar seus cálculos. Apenas um operador aritmético é utilizado por estágio e multiplexadores são usados para selecionar as entradas corretas dos operadores a cada novo ciclo.

As arquiteturas unidimensionais utilizam *buffers* tipo ping-pong para manter estáveis as quatro entradas dos multiplexadores nos *buffers* pong (registradores em branco na Figura 4.1) enquanto novas entradas seriais continuam a chegar e vão sendo armazenadas nos *buffers* ping (registradores em cinza na Figura 4.1). Esta estratégia é adotada para viabilizar o sincronismo no pipeline.

As duas arquiteturas em uma dimensão são unidas por um *buffer* de transposição. Este *buffer* é responsável por ler linha a linha os resultados da primeira Hadamard 1-D e entregar estes resultados coluna a coluna para a segunda Hadamard 1-D.

O *buffer* de transposição foi implementado utilizando duas pequenas memórias RAM. Cada memória possui 16 posições com 16 bits em cada posição, para armazenar os 16 resultados da primeira Hadamard 1-D. As duas memórias funcionam de maneira intercalada, isto é, enquanto os resultados da primeira Hadamard 1-D são escritos em uma memória, a outra memória está provendo os dados para a segunda Hadamard 1-D. Quando a operação de escrita e leitura de um bloco acaba, então as memórias mudam de operação, isso é, a memória que estava sendo lida não possui mais nenhum dado que não tenha sido utilizado e, então, novos valores são escritos nesta memória. Por outro lado, a memória que acabou de ser preenchida com novos valores provenientes da primeira Hadamard 1-D, inicia a operação de prover estes novos dados para a segunda Hadamard 1-D.

A solução serial e com separabilidade também foi desenvolvida para a FDCT 2-D e para a IDCT 2-D, pois a idéia inicial era usar esta estratégia no desenvolvimento do módulo T completo. Após analisar os resultados das demais estratégias de implementação das transformadas, a solução serial com separabilidade acabou sendo descartada, por apresentar resultados inferiores aos de outras soluções, como será explicado nas próximas seções. De qualquer modo, os resultados gerados pelo projeto destas duas transformadas foram publicados em (AGOSTINI, 2007).

4.1.2 Arquitetura Serial sem Separabilidade

A segunda alternativa arquitetural desenvolvida considerou uma arquitetura que ainda consumisse uma amostra a cada ciclo (serial, portanto), mas que realizasse diretamente o cálculo da Hadamard 2-D 4x4 em duas dimensões (sem separabilidade). Esta solução está apresentada na Figura 4.2.

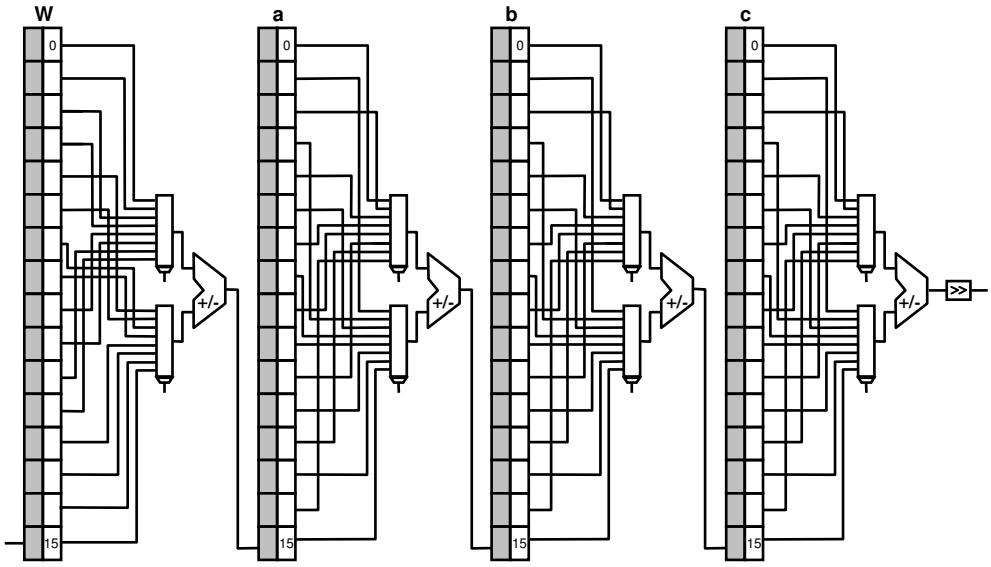

Figura 4.2: Arquitetura da transformada serial sem separabilidade

Para tornar esta solução viável, a fórmula de cálculo da Hadamard 2-D que foi apresentada em (13) foi desdobrada, gerando o algoritmo apresentado na Tabela 4.1. Estes cálculos geram um resultado que é idêntico ao produzido pela fórmula (13), mas sem usar a separabilidade.

Tabela 4.1: Algoritmo utilizado para o cálculo da Hadamard 2-D 4x4 direta

$a_0 = W_0 + W_4$	$b_0 = a_0 + a_1$	$c_0 = b_0 + b_1$	$S_0 = (c_0 + c_1)/2$
$a_1 = W_8 + W_{12}$	$b_1 = a_2 + a_3$	$c_1 = b_2 + b_3$	$S_1 = (c_0 - c_1)/2$
$a_2 = W_1 + W_5$	$b_2 = a_4 + a_5$	$c_2 = b_0 - b_1$	$S_2 = (c_2 - c_3)/2$
$a_3 = W_9 + W_{13}$	$b_3 = a_6 + a_7$	$c_3 = b_2 - b_3$	$S_3 = (c_2 + c_3)/2$
$a_4 = W_2 + W_6$	$b_4 = a_0 - a_1$	$c_4 = b_4 + b_5$	$S_4 = (c_4 + c_5)/2$
$a_5 = W_{10} + W_{14}$	$b_5 = a_2 - a_3$	$c_5 = b_6 + b_7$	$S_5 = (c_4 - c_5)/2$
$a_6 = W_3 + W_7$	$b_6 = a_4 - a_5$	$c_6 = b_4 - b_5$	$S_6 = (c_6 - c_7)/2$
$a_7 = W_{11} + W_{15}$	$b_7 = a_6 - a_7$	$c_7 = b_6 - b_7$	$S_7 = (c_6 + c_7)/2$
$a_8 = W_0 - W_4$	$b_8 = a_8 - a_9$	$c_8 = b_8 + b_9$	$S_8 = (c_8 + c_9)/2$
$a_9 = W_8 - W_{12}$	$b_9 = a_{10} - a_{11}$	$c_9 = b_{10} + b_{11}$	$S_9 = (c_8 - c_9)/2$
$a_{10} = W_1 - W_5$	$b_{10} = a_{12} - a_{13}$	$c_{10} = b_8 - b_9$	$S_{10} = (c_{10} - c_{11})/2$
$a_{11} = W_9 - W_{13}$	$b_{11} = a_{14} - a_{15}$	$c_{11} = b_{10} - b_{11}$	$S_{11} = (c_{10} + c_{11})/2$
$a_{12} = W_2 - W_6$	$b_{12} = a_8 + a_9$	$c_{12} = b_{12} + b_{13}$	$S_{12} = (c_{12} + c_{13})/2$
$a_{13} = W_{10} - W_{14}$	$b_{13} = a_{10} + a_{11}$	$c_{13} = b_{14} + b_{15}$	$S_{13} = (c_{12} - c_{13})/2$
$a_{14} = W_3 - W_7$	$b_{14} = a_{12} + a_{13}$	$c_{14} = b_{12} - b_{13}$	$S_{14} = (c_{14} - c_{15})/2$
$a_{15} = W_{11} - W_{15}$	$b_{15} = a_{14} + a_{15}$	$c_{15} = b_{14} - b_{15}$	$S_{15} = (c_{14} + c_{15})/2$

O impacto do não uso da separabilidade é notável já pelo algoritmo apresentado na Tabela 4.1, que é visivelmente mais complexo do que a fórmula (13). A decisão de não utilizar a separabilidade causa um aumento na complexidade da arquitetura para o cálculo da Hadamard 2-D 4x4, aumentando a utilização de recursos necessários para a sua implementação em hardware. Em contrapartida, o *buffer* de transposição não é mais necessário e, então, esta arquitetura não utiliza bits de memória RAM, usando apenas registradores como elementos de armazenamento. Essa é uma característica importante quando o projeto do codec H.264/AVC como um todo é levado em consideração, pois

alguns módulos do codec precisam utilizar muita memória RAM. Como os dispositivos alvo deste trabalho são FPGAs, que possuem restritos bits de memória RAM interna, toda a economia de bits de memória é bem vinda. Outra importante vantagem do uso do algoritmo da Tabela 4.1, sem separabilidade e, por conseqüência, do uso desta arquitetura, é que ela pode ser facilmente paralelizada. Adaptar esta arquitetura para um paralelismo de dois, quatro, oito ou dezesseis dados sendo consumidos a cada ciclo é possível e é uma tarefa pouco complexa. Para aumentar o paralelismo basta aumentar o número de operadores por estágio do *pipeline*. No extremo, se dezesseis operadores são usados em cada estágio, então uma matriz 4x4 completa pode ser processada a cada ciclo.

A arquitetura desenvolvida para a solução serial sem separabilidade também utiliza *buffers* do tipo ping-pong. Neste caso, os *buffers* ping-pong irão possuir 16 posições, para armazenar toda uma matriz 4x4 de entrada. A arquitetura sem separabilidade foi projetada em um *pipeline* de quatro estágios, sendo que cada estágio utiliza 16 ciclos de *clock* para finalizar as suas operações. Novamente apenas um operador é utilizado em cada estágio de *pipeline*. A latência da arquitetura sem separabilidade é de 64 ciclos de *clock*.

4.1.3 Arquitetura Parcialmente Paralela

A terceira alternativa arquitetural investigada foi uma arquitetura parcialmente paralela, que consome uma linha do bloco 4x4 de entrada a cada ciclo, isto é, são consumidas quatro amostras por ciclo de *clock*. Esta solução está apresentada na Figura 4.3.

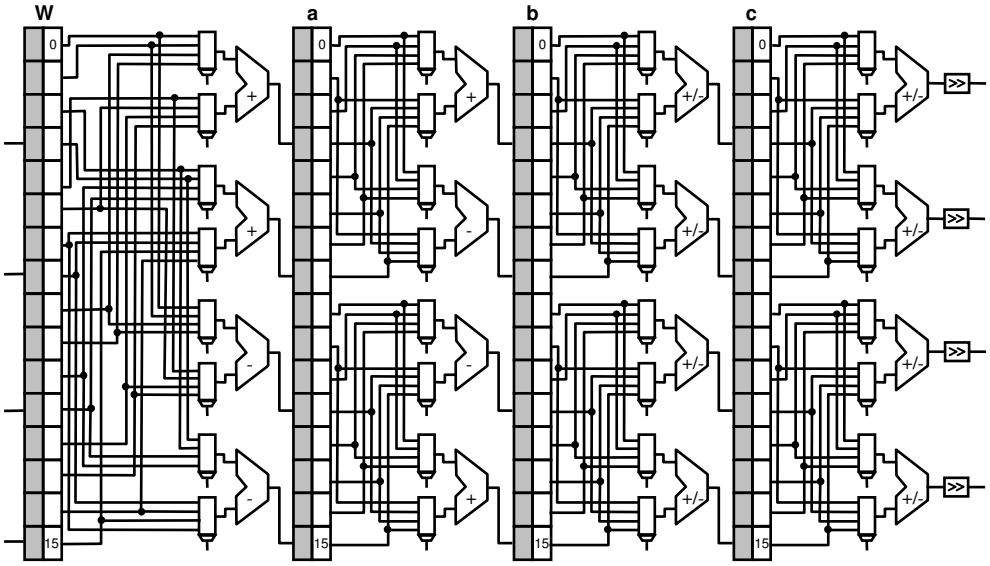

Figura 4.3: Arquitetura da transformada paralela com quatro leituras por ciclo

A arquitetura desenvolvida para a solução parcialmente paralela utiliza o mesmo algoritmo da solução serial sem separabilidade, apresentado na Tabela 4.1. A arquitetura parcialmente paralela recebe quatro amostras de entrada a cada ciclo e, com o *pipeline* cheio, entrega também quatro valores por ciclo na saída.

Também foram usados *buffers* do tipo ping-pong nesta arquitetura. O número de registradores utilizados é o mesmo que para a solução serial sem separabilidade, mas a localidade dos dados permitiu agrupar alguns registradores de modo a permitir a divisão de alguns dos *buffers* de 16 posições em *buffers* menores, com oito posições. Outra

constatação importante é que todos os *buffers* ping-pong desta arquitetura recebem e entregam mais de um valor a cada ciclo de *clock*, diferentemente do que ocorre nos *buffers* ping-pong das arquiteturas anteriormente apresentadas, que recebem e entregam apenas um valor a cada ciclo. A arquitetura parcialmente paralela, do mesmo modo que a arquitetura serial sem separabilidade, foi projetada em um *pipeline* de quatro estágios, sendo que cada estágio, neste caso, utiliza quatro ciclos de *clock* para finalizar as suas operações. A arquitetura serial sem separabilidade utiliza 16 ciclos em cada estágio. Na arquitetura parcialmente paralela, como pode ser observado na Figura 4.3, são utilizados quatro operadores em cada estágio do *pipeline*, permitindo esta redução no número de ciclos utilizado por estágio. A latência também diminui em função do paralelismo, mesmo que parcial, ficando em 16 ciclos de *clock*, contra os 64 ciclos da arquitetura serial sem separabilidade.

4.1.4 Arquitetura Paralela com *Pipeline*

A quarta alternativa arquitetural explorada foi uma arquitetura completamente paralela, consumindo 16 amostras (um bloco completo de 4x4 amostras) a cada ciclo de *clock*. Esta arquitetura também é baseada no algoritmo apresentado na Tabela 4.1 e não utiliza a propriedade da separabilidade. Esta solução está apresentada na Figura 4.4 e foi desenvolvida em um *pipeline* de quatro estágios, do mesmo modo que as arquiteturas serial sem separabilidade e parcialmente paralela. A diferença é que, na arquitetura completamente paralela, apenas um ciclo é consumido em cada estágio do *pipeline*, contra os dezesseis ciclos utilizados pela arquitetura serial e os quatro ciclos utilizados pela arquitetura parcialmente paralela. Deste modo, a latência da arquitetura paralela é de apenas quatro ciclos de *clock*. A taxa de processamento desta arquitetura é de 16 amostras por ciclo.

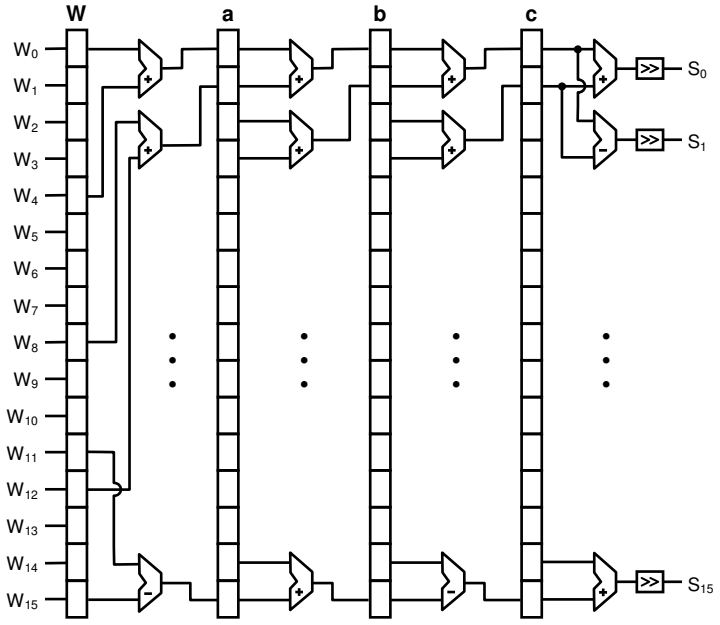

Figura 4.4: Arquitetura da transformada paralela em *pipeline*

Existem dois problemas principais com esta solução. A primeira é que são necessários 16 operadores por estágio de *pipeline*, o que conduz a um consumo elevado de recursos. A segunda diz respeito ao roteamento, pois são 16 amostras entrando e 16 amostras saindo, conduzindo a um número de fios muito grande entrando e saindo da arquitetura. A grande vantagem está na enorme taxa de processamento que esta

arquitetura pode atingir, mas também é importante salientar outras duas vantagens. A primeira diz respeito ao projeto da parte de controle que é extremamente simples para a arquitetura completamente paralela. A segunda é que esta arquitetura não precisa utilizar os *buffers* ping-pong utilizados nas soluções anteriores, pois registradores simples são suficientes. A substituição dos *buffers* ping-pong por registradores simples reduz a área utilizada e aumenta o desempenho da arquitetura.

4.1.5 Arquitetura Paralela Combinacional

A quinta e última alternativa arquitetural investigada foi uma arquitetura paralela completamente combinacional, também consumindo 16 amostras por ciclo, como a arquitetura paralela em *pipeline*. Esta solução está apresentada na Figura 4.5.

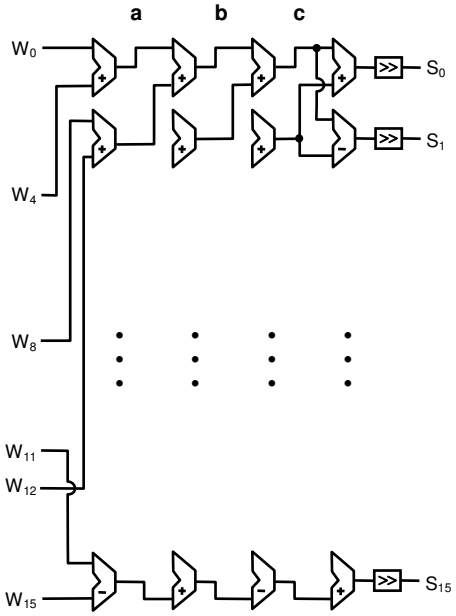

Figura 4.5: Arquitetura da transformada paralela combinacional

Esta arquitetura, por ser puramente combinacional, não utiliza registradores, utilizando uma quantidade menor de recursos de hardware para ser implementada. Mas a ausência de registradores também conduz a uma desvantagem que é o aumento do atraso. Enquanto o caminho crítico das demais arquiteturas apresentadas até agora era de um registrador e um somador, na arquitetura combinacional este caminho crítico vai passar por quatro somadores em série.

Outra vantagem é a inexistência da parte de controle, facilitando o projeto e também reduzindo o consumo de recursos de hardware.

A arquitetura puramente combinacional possui uma latência de apenas um ciclo, contra as latências mais elevadas das demais arquiteturas apresentadas, que podem chegar até a 64 ciclos de *clock*.

4.1.6 Resultados de Síntese

Os resultados de síntese das arquiteturas apresentadas nas seções anteriores estão resumidos na Tabela 4.2. Nestas sínteses, o dispositivo alvo foi um VP70 da família Virtex-II Pro (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada foi o Synplify Pro da Synplicity (SYNPLICITY, 2007).

Tabela 4.2: Resultados comparativos entre as diversas alternativas arquiteturais para a transformada Hadamard 4x4 direta

Projeto	Elementos Lógicos	Freqüência (MHz)	Bits de Memória	Taxa de Processamento (Mamostras/s)	Latência (ciclos)
Serial com Separabilidade	270	190,3	320	190,3	24
Serial sem Separabilidade	609	152,4	0	152,4	64
Parcialmente Paralela	801	330,0	0	1320,0	16
Paralela com Pipeline	1152	259,7	0	4155,2	4
Paralela Combinacional	1140	71,2	0	1139,2	1

Dispositivo 2VP70FF1517-6

É importante destacar que a Tabela 4.2 não apresenta os resultados de consumo de registradores para estas arquiteturas. As arquiteturas que mais usam registradores são aquelas que possuem *buffer ping-pong*: serial com separabilidade, serial sem separabilidade e parcialmente paralela. Depois vem a arquitetura paralela com pipeline, que usa linhas de registradores simples. Neste quesito, o melhor desempenho é atingido pela solução paralela combinacional, que não usa nenhum registrador.

Como pode ser percebido através da Tabela 4.2, a arquitetura que utilizou menos elementos lógicos foi a arquitetura da transformada Hadamard serial com separabilidade, como já era esperado. Por outro lado, esta é a única arquitetura que utiliza bits de memória.

A arquitetura com maior freqüência de operação é a parcialmente paralela, atingindo a segunda maior taxa de processamento, perdendo apenas para a arquitetura paralela com pipeline. Esta arquitetura é a terceira com maior consumo de elementos lógicos, perdendo apenas para a arquitetura paralela com pipeline e para a arquitetura parcialmente paralela.

A arquitetura paralela com pipeline é a que utiliza a maior quantidade de elementos lógicos. Esta arquitetura possui a segunda freqüência mais elevada dentre as implementações apresentadas, perdendo apenas para a arquitetura parcialmente paralela, sendo capaz de suportar a mais elevada taxa de processamento, chegando a 4,1 bilhões de amostras por segundo. Mas o maior problema desta solução é que ela exige um elevado número de conexões de entrada e saída, dificultando sua inserção no compressor H.264/AVC.

A arquitetura paralela combinacional é a que apresenta a mais baixa freqüência de operação dentre todas as soluções apresentadas, mas, mesmo assim, consegue atingir a terceira maior taxa de processamento, perdendo apenas para a arquitetura paralela em *pipeline* e para a arquitetura parcialmente paralela. Além disso, a arquitetura paralela combinacional é a segunda que mais utiliza elementos lógicos, perdendo apenas para a arquitetura paralela em pipeline.

A transformada serial sem separabilidade é a que possui a pior taxa de processamento, mas é a segunda que menos consome elementos lógicos. A maior vantagem desta arquitetura é que ela consome e produz apenas uma amostra por ciclo, como a arquitetura serial com separabilidade, e, deste modo, é facilmente integrável aos

demais módulos de um codec H.264/AVC. Por outro lado, esta arquitetura, ao contrário do que acontece com a arquitetura serial com separabilidade, não utiliza bits de memória RAM, o que é uma característica desejável quando se considera o projeto do codec H.264/AVC como um todo. Por conta destas características, esta foi a alternativa escolhida para implementar a primeira versão das arquiteturas de transformadas utilizadas nos módulos T e T^{-1}, cujos projetos estão apresentados nas seções 4.2 e 4.4 do texto.

É importante salientar que a taxa de processamento de todas as transformadas apresentadas na Tabela 4.2, mesmo variando de 152 milhões a 4,1 bilhões de amostras por segundo, é suficiente, inclusive, para sua utilização em codecs H.264/AVC para televisão digital de alta resolução (HDTV). Por exemplo, considerando um quadro HDTV de 1920x1080 pixels sendo gerado a 30 quadros por segundo e com um espaço de cores com luminância e crominância em uma relação de 4:2:0 (como está previsto pelos modos *baseline*, *main* e *extended* do padrão H.264/AVC), então, no pior caso, as transformadas deveriam suprir uma taxa de processamento de 93,3 Mamostras/s.

Os resultados obtidos neste experimento foram publicados em (PORTO, 2005). Existem alguns outros trabalhos publicados relatando arquiteturas desenvolvidas para as transformadas do padrão H.264/AVC. A maioria destes trabalhos se refere a soluções que processam mais de uma transformada com o mesmo hardware, mas em tempos distintos. Apenas o trabalho desenvolvido por Kordasiewicz (2004) apresenta uma solução que processa somente uma transformada. Esta solução é completamente paralela e foca apenas na DCT 2-D. A arquitetura atinge uma taxa de processamento de 1.720 amostras por segundo, com um desempenho quase 2,5 vezes menor do que a solução paralela com pipeline desenvolvida neste trabalho e que possui o mesmo nível de paralelismo.

4.2 Arquitetura Serial para o Módulo das Transformadas Diretas

Em função da investigação apresentada na seção 4.1 deste trabalho e da decisão pelo uso das transformadas seriais sem separabilidade para a primeira implementação do módulo T, as arquiteturas da FDCT 2-D e da Hadamard 2-D 2x2 direta foram construídas de acordo com a estratégia serial sem separabilidade. É importante lembrar que a Hadamard 2-D 4x4 direta foi desenvolvida na seção 4.1.2. Esta seção irá apresentar os algoritmos e as arquiteturas desenvolvidas para as transformadas diretas utilizando a estratégia serial sem separabilidade, bem como o projeto completo do módulo T. O projeto do módulo T buscou a facilidade de integração deste módulo com os demais módulos do codificador H.264/AVC. Por isso, foi construída uma arquitetura que consome e produz uma amostra a cada ciclo de *clock*, através da sincronização dos diferentes caminhos de dados presentes no interior do módulo T.

Os resultados das transformadas diretas, bem como do projeto do módulo T como um todo estão apresentados nas próximas seções.

4.2.1 A Arquitetura da FDCT 2-D Serial

O algoritmo utilizado para o cálculo da FDCT 2-D está apresentado na Tabela 4.3 e foi extraído da definição apresentada em (11). Este algoritmo realiza o cálculo da FDCT 2-D de maneira direta, isto é, não utiliza a propriedade da separabilidade.

Tabela 4.3: Algoritmo utilizado para o cálculo da FDCT 2-D

$a_0 = X_0 + X_{12}$	$b_0 = a_0 + a_1$	$c_0 = b_0 + b_3$	$S_0 = c_0 + c_1$
$a_1 = X_4 + X_8$	$b_1 = a_2 + a_3$	$c_1 = b_1 + b_2$	$S_1 = 2*c_2 + c_3$
$a_2 = X_1 + X_{13}$	$b_2 = a_4 + a_5$	$c_2 = b_0 - b_3$	$S_2 = c_0 - c_1$
$a_3 = X_5 + X_9$	$b_3 = a_6 + a_7$	$c_3 = b_1 - b_2$	$S_3 = c_2 - 2*c_3$
$a_4 = X_2 + X_{14}$	$b_4 = a_8 + a_{14}$	$c_4 = b_4 + b_6$	$S_4 = 2*c_4 + c_5$
$a_5 = X_6 + X_{10}$	$b_5 = a_9 + a_{15}$	$c_5 = b_5 + b_7$	$S_5 = 2*c_8 + c_9$
$a_6 = X_3 + X_{15}$	$b_6 = a_{10} + a_{12}$	$c_6 = b_4 - b_6$	$S_6 = 2*c_6 + c_7$
$a_7 = X_7 + X_{11}$	$b_7 = a_{11} + a_{13}$	$c_7 = b_5 - b_7$	$S_7 = c_8 - 2*c_9$
$a_8 = X_0 - X_{12}$	$b_8 = a_8 - a_{14}$	$c_8 = 2*b_8 + b_9$	$S_8 = c_{12} + c_{13}$
$a_9 = X_4 - X_8$	$b_9 = a_9 - a_{15}$	$c_9 = 2*b_{10} + b_{11}$	$S_9 = 2*c_{14} + c_{15}$
$a_{10} = X_1 - X_{13}$	$b_{10} = a_{10} - a_{12}$	$c_{10} = b_8 - 2*b_9$	$S_{10} = c_{12} - c_{13}$
$a_{11} = X_5 - X_9$	$b_{11} = a_{11} - a_{13}$	$c_{11} = b_{10} - 2*b_{11}$	$S_{11} = c_{14} - 2*c_{15}$
$a_{12} = X_2 - X_{14}$	$b_{12} = a_0 - a_1$	$c_{12} = b_{12} + b_{15}$	$S_{12} = c_4 - 2*c_5$
$a_{13} = X_6 - X_{10}$	$b_{13} = a_2 - a_3$	$c_{13} = b_{13} + b_{14}$	$S_{13} = 2*c_{10} + c_{11}$
$a_{14} = X_3 - X_{15}$	$b_{14} = a_4 - a_5$	$c_{14} = b_{12} - b_{15}$	$S_{14} = c_6 - 2*c_7$
$a_{15} = X_7 - X_{11}$	$b_{15} = a_6 - a_7$	$c_{15} = b_{13} - b_{14}$	$S_{15} = c_{10} - 2*c_{11}$

Na Tabela 4.3, $\mathbf{a_0}$ a $\mathbf{a_{15}}$, $\mathbf{b_0}$ a $\mathbf{b_{15}}$ e $\mathbf{c_0}$ a $\mathbf{c_{15}}$ são variáveis internas, que são úteis à implementação em *pipeline* que foi desenvolvida. As saídas são representadas pelas variáveis $\mathbf{S_0}$ a $\mathbf{S_{15}}$ e as entradas são as variáveis $\mathbf{X_0}$ a $\mathbf{X_{15}}$.

A arquitetura desenvolvida para implementar a FDCT 2-D, segundo o algoritmo apresentado na Tabela 4.3, está apresentada na Figura 4.6. Esta arquitetura é muito parecida com aquela apresentada na Figura 4.2 para a transformada Hadamard 2-D 4x4 direta. A arquitetura da FDCT 2-D sem separabilidade foi projetada em um *pipeline* de quatro estágios, sendo que cada estágio utiliza 16 ciclos de *clock* para finalizar as suas operações. Novamente, apenas um operador é utilizado por estágio de *pipeline*. A latência da arquitetura sem separabilidade é de 64 ciclos de *clock* e são utilizados 8 bits na entrada e 14 bits na saída.

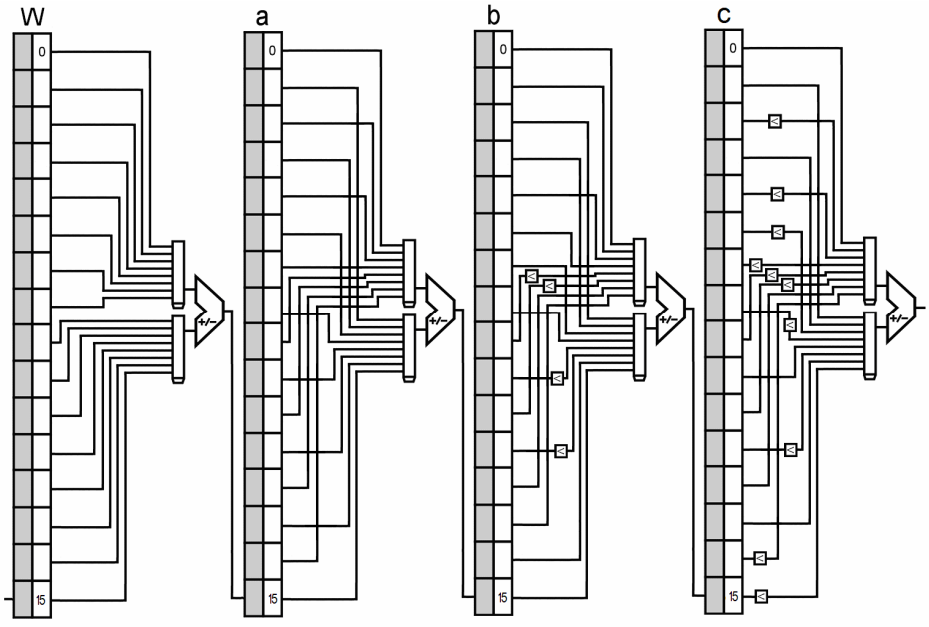

Figura 4.6: Arquitetura serial sem separabilidade para o cálculo da FDCT 2-D

4.2.2 A Arquitetura da Hadamard 2-D 2x2 Direta Serial

O algoritmo utilizado para o cálculo da Hadamard 2x2 direta ou inversa é o mesmo e está apresentado na Tabela 4.4, tendo sido extraído da definição apresentada em (14). Na Tabela 4.4, $\mathbf{a_0}$ a $\mathbf{a_3}$ são variáveis internas, que são úteis à implementação em *pipeline*. As saídas são representadas pelas variáveis $\mathbf{S_0}$ a $\mathbf{S_3}$ e a entrada são as variáveis $\mathbf{W_0}$ a $\mathbf{W_3}$.

Tabela 4.4: Algoritmo utilizado para o cálculo da Hadamard 2-D 2x2 direta ou inversa

$a_0 = W_0 + W_2$	$S_0 = a_0 + a_2$
$a_1 = W_0 - W_2$	$S_1 = a_0 - a_2$
$a_2 = W_3 + W_1$	$S_2 = a_1 - a_3$
$a_3 = W_3 - W_1$	$S_3 = a_1 + a_3$

A arquitetura desenvolvida para implementar a Hadamard 2-D 2x2 (direta e inversa) é serial, como no caso das demais transformadas, ou seja, processa uma amostra a cada ciclo de *clock*. Além disso, esta arquitetura foi desenvolvida em um *pipeline* de dois estágios, para aumentar a taxa de processamento. Quando o *pipeline* está cheio, a arquitetura entrega um coeficiente válido a cada ciclo. Novamente apenas um operador é utilizado por estágio de *pipeline*. A latência da arquitetura sem separabilidade é de 8 ciclos de *clock*. A implementação da Hadamard 2-D 2x2 utiliza 14 bits de entrada e 16 bits de saída. Esta arquitetura está apresentada na Figura 4.7.

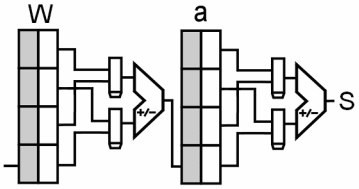

Figura 4.7: Arquitetura serial sem separabilidade para cálculo da Hadamard 2-D 2x2 direta e inversa

4.2.3 Arquitetura do Módulo T Serial

O diagrama de blocos do módulo T está apresentado na Figura 4.8. Além das arquiteturas das transformadas (FDCT 4x4, Hadamard 4x4 Direta e Hadamard 2x2 Direta na Figura 4.8), foram incluídos alguns *buffers* para armazenar o resultado parcial dos cálculos e para sincronizar as operações realizadas, no intuito de facilitar a integração deste módulo aos demais módulos do compressor.

A arquitetura proposta para o módulo T tem por objetivo sincronizar as operações apresentadas na seção 3.6.4 deste texto, de modo a consumirem uma amostra em cada ciclo de *clock* e, quando o pipeline está cheio, gerar um valor a cada ciclo de *clock*. Por isso, a FIFO foi inserida no caminho da FDCT 2-D para atrasar sua saída quando o modo é não intra-quadro 16x16 e, deste modo, sincronizar as operações. Este sincronismo facilita muito a integração desta arquitetura aos demais módulos do compressor, pois após o *pipeline* estar cheio, um valor válido é gerado pelo módulo T a cada ciclo de *clock*, não importando o modo de previsão que foi adotado.

Além da FIFO no caminho da FDCT 2-D, outros *buffers* foram necessários. Foram usados dois *buffers* para armazenar os componentes DC de luminância e crominância

(*buffer* Y_{DC} e *buffer* C_{DC} na Figura 4.8) que serão necessários para o cálculo das Hadamard (quando elas são utilizadas). Outros dois *buffers* foram usados para armazenar os valores AC de luminância e crominância (*buffer* Y_{AC} e *buffer* C_{AC} na Figura 4.8) enquanto as transformadas Hadamard finalizam seus cálculos. Estes últimos *buffers* são necessários para garantir a ordem correta dos elementos na saída.

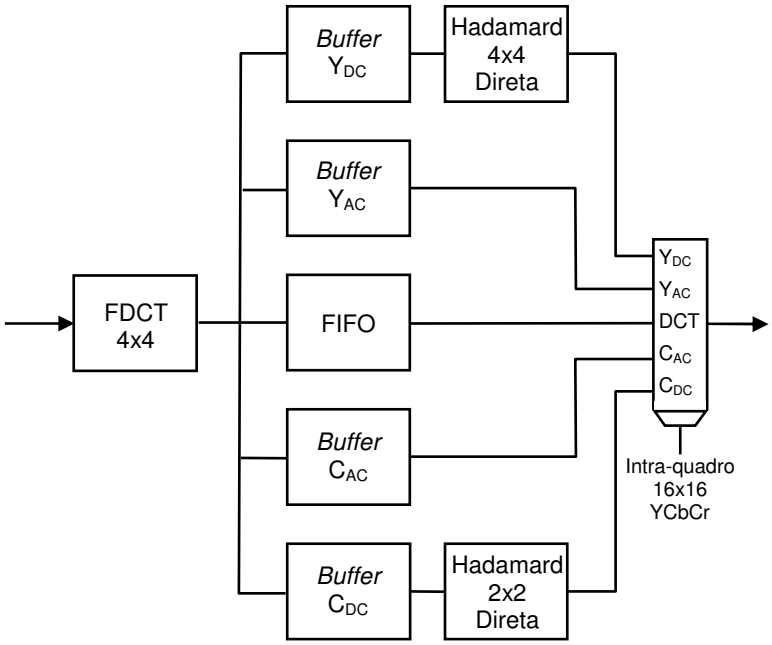

Figura 4.8: Diagrama em blocos da arquitetura do módulo T serial

Todos os caminhos de dados que passam pelo módulo T possuem a mesma latência (384 ciclos de *clock*) e a geração de um valor por ciclo de *clock* na saída pode ser garantida se a taxa de entrada for, também, de uma amostra por ciclo. A entrada do módulo T é o resíduo gerado pelo modo de predição escolhido e a saída é entregue para a quantização (módulo Q).

Os *buffers* e a FIFO do módulo T foram desenvolvidas utilizando registradores ao invés de memória. Assim, foi possível eliminar o uso de memória RAM neste módulo e liberar memória para onde ela é mais necessária nos demais módulos do codec H.264/AVC.

O módulo T utiliza 8 bits para cada resíduo de entrada e gera 17 bits para cada coeficiente na sua saída. Este incremento na faixa dinâmica é causado pelas operações de somas e subtrações que geram *carry out* no interior dos módulos das transformadas.

A parte de controle do módulo T é responsável pela sincronização entre as operações dos elementos formadores do módulo. Especial destaque foi dado à gerência de escritas nos *buffers* e na FIFO, pois nenhum dado poderia ser perdido e nenhum ciclo de *clock* devia ser desperdiçado. O controle foi projetado através de máquinas de estados finitos que interagem para dividir a tarefa de gerar os sinais de controle, a partir dos sinais externos e do dado que está sendo processado. O controle entrega à parte operativa os sinais de controle dos multiplexadores e de habilitação de escrita nos *buffers*. Estes sinais indicam se a amostra que está sendo processada é AC ou DC, se ela foi codificada no modo intra-quadro 16x16 e se a amostra é Y, Cb ou Cr.

4.2.4 Resultados de Síntese do Módulo T Serial

A Tabela 4.5 apresenta os resultados de síntese dos principais componentes formadores do módulo T, bem como do próprio módulo T. Novamente, o dispositivo alvo foi um FPGA Virtex-II Pro VP70 (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada foi o Synplify Pro da Synplicity (SYNPLICITY, 2007).

Tabela 4.5: Resultados de síntese do módulo T serial

Transformada	Elementos Lógicos	Freqüência (MHz)	Taxa de Processamento (Mamostras/s)
Controle	809	191,7	-
FDCT 4x4 Serial	475	155,3	155,3
Hadamard 4x4 Direta Serial	609	152,4	152,4
Hadamard 2x2 Direta Serial	98	201,7	201,7
FIFO	256	208,2	208,2
***Buffer* Y_{DC}**	31	318,1	318,1
***Buffer* Y_{AC}**	361	207,4	207,4
***Buffer* C_{DC}**	31	318,1	318,1
***Buffer* C_{AC}**	76	320,6	320,6
Módulo T Serial	2359	138,7	138,7

Dispositivo 2VP70FF1517-6

O resultado mais importante apresentado na Tabela 4.5 é relacionado à máxima taxa de processamento dos componentes internos ao módulo T que, para todos os casos, é suficiente para a utilização em codecs H.264/AVC para HDTV, pois suas taxas de processamento são superiores a 93,3 milhões de amostras por segundo. A maior freqüência de operação foi encontrada para o *buffer* C_{AC}, que atingiu 320MHz, permitindo uma taxa de processamento superior a 320 milhões de amostras por segundo. Por outro lado, a freqüência mais baixa foi encontrada para a Hadamard 4x4 direta, que atingiu 152,4 MHz, permitindo uma taxa de processamento superior a 152 milhões de amostras por segundo. Mesmo o bloco de menor freqüência pode ser utilizado em codecs voltados para HDTV.

Conforme os resultados da Tabela 4.5, a arquitetura da Hadamard 2-D 4x4 direta utiliza mais elementos lógicos do que a arquitetura da FDCT 2-D. Isso ocorre porque a largura de bits de entrada desta arquitetura é mais elevada do que a largura de bits da arquitetura da FDCT 2-D. A mesma justificativa é válida para a freqüência de operação atingida ser superior na FDT 2-D do que na Hadamard 2-D 4x4.

O módulo do controle foi o maior responsável pelo consumo de elementos lógicos do FPGA, seguido pelos módulos das transformadas 4x4. Todos os módulos juntos utilizam menos de 3% dos elementos lógicos disponíveis no FPGA alvo.

O módulo T, por sua vez, também atingiu os requisitos de HDTV, pois atingiu 138,7MHz, permitindo uma taxa de processamento superior a 138 milhões de amostras por segundo. Deste modo, a implementação do módulo T apresentada neste trabalho

possui uma folga mínima de desempenho superior a 45 milhões de amostras por segundo se o projeto do codec H.264/AVC exigir desempenho de HDTV. Deste modo, como a resolução de HDTV é uma das mais elevadas na atualidade, qualquer codec H.264/AVC direcionado para os padrões de menor resolução poderá utilizar a arquitetura do módulo T desenvolvida e apresentada neste trabalho. Do ponto de vista do consumo de elementos lógicos do FPGA alvo, esta implementação utilizou cerca de 3% do total de elementos lógicos disponíveis, em um total de 2.359 elementos lógicos utilizados.

Esta arquitetura foi publicada em (AGOSTINI, 2006b) e nenhuma outra arquitetura para o módulo T completo foi encontrada na literatura.

4.3 Arquitetura Paralela para o Módulo das Transformadas Diretas

Após o desenvolvimento do módulo das transformadas diretas de maneira serial, foi constatada a importância do desenvolvimento de uma solução para as transformadas e quantização diretas e inversas com máxima taxa de processamento, em função da dependência dos dados do quadro atual reconstruído por parte da predição intra-quadro. Então, a primeira solução nesta direção foi a construção de uma arquitetura completamente paralela para o módulo das transformadas diretas. Esta arquitetura está relatada nesta seção e é capaz de consumir e produzir 16 amostras por ciclo.

Os algoritmos utilizados para o cálculo das transformadas na implementação paralela do módulo T foram os mesmos apresentados nas seções 4.1.2, 4.2.1 e 4.2.2. As arquiteturas foram embasadas na solução paralela apresentada na seção 4.1.4.

4.3.1 Arquitetura das Transformadas Paralelas

As transformadas foram desenvolvidas em arquiteturas totalmente paralelas. Dessa forma, a FDCT e a Hadamard 4x4 direta consomem 16 amostras a cada ciclo de *clock*, enquanto que a Hadamard 2x2 direta consome 4 amostras por ciclo de *clock*.

As arquiteturas da FDCT 2-D e da Hadamard 4x4 direta são similares, tendo sido implementadas com 4 estágios de pipeline, onde cada estágio possui 16 operadores. A latência de ambas as arquiteturas é de quatro ciclos de *clock*. A arquitetura paralela da Hadamard 4x4 direta foi apresentada na seção 4.1.4 e não será apresentada novamente. A arquitetura da FDCT é similar à da Hadamard 4x4 direta e também não será apresentada. As diferenças destas arquiteturas estão nas entradas dos 16 operadores de cada estágio de pipeline e nos deslocamentos adicionais presentes em algumas entradas destes operadores. Mas a principal diferença está na faixa dinâmica das duas soluções, pois, enquanto da FDCT 2-D recebe entradas com 8 bits por amostra e gera saídas com 12 bits por amostra, a Hadamard 2-D direta possui entradas com 12 bits por amostra e saídas com 15 bits por amostra.

A arquitetura paralela da Hadamard 2x2 direta foi desenvolvida em um pipeline de dois estágios, com quatro operadores por estágio. Esta arquitetura está apresentada na Figura 4.9 e possui uma latência de dois ciclos de *clock*. A Hadamard 2x2 recebe amostras com 12 bits e gera amostras com 14 bits.

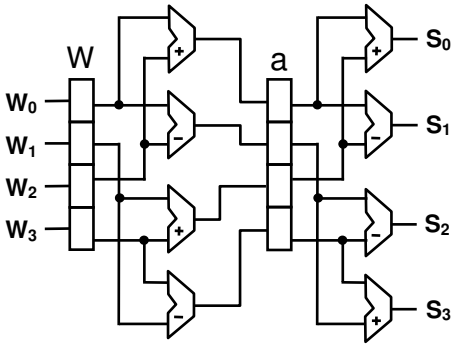

Figura 4.9: Arquitetura paralela da Hadamard 2x2

4.3.2 A Arquitetura do Módulo T Paralelo

A arquitetura do módulo T paralelo reúne as arquiteturas das três transformadas paralelas descritas na seção anterior. Este módulo é responsável, também, pelo sincronismo dos dados de entrada e saída, como já foi explicado na seção 4.2 e, por isso, a inserção de alguns *buffers* se faz necessária. A Figura 4.10 ilustra o diagrama de blocos da arquitetura desenvolvida para o módulo T paralelo. O módulo T é formado pelos blocos destacados com o fundo cinza na Figura 4.10, onde também são apresentados os módulos paralelos da quantização, que não fazem parte do escopo da arquitetura do bloco T paralelo.

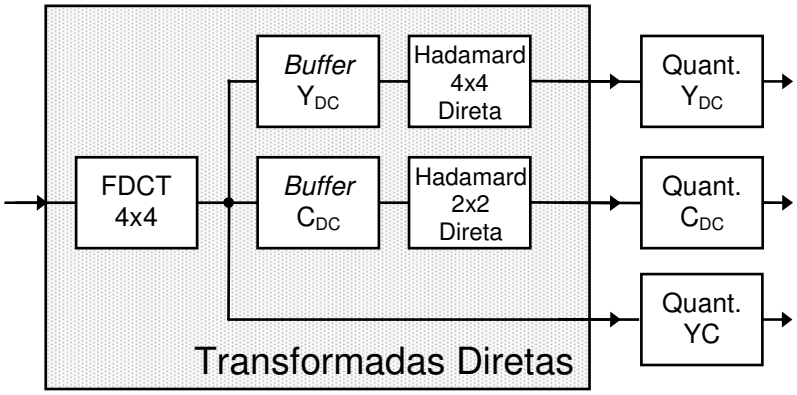

Figura 4.10: Diagrama em blocos da arquitetura do módulo T paralelo

Uma das principais vantagens da implementação paralela do módulo T é a drástica simplificação do controle. Na versão serial, o controle é bastante complexo, pois é necessário manter o sincronismo de escrita e leitura nos diversos *buffers* presentes na arquitetura. Já na versão totalmente paralela, como pode ser observado na Figura 4.10, apenas dois *buffers* são necessários para sincronizar os dados de entrada das Hadamard 4x4 e 2x2. Os resultados do módulo T são disponibilizados em 3 saídas paralelas, duas com 16 amostras e uma com quatro amostras. Então, para viabilizar a utilização de uma implementação completamente paralela para o módulo das transformadas diretas, devem ser desenvolvidos três módulos paralelos para o cálculo da quantização, como está apresentado na Figura 4.10.

A FDCT 4x4 é a primeira operação deste módulo e é comum para todas as amostras de entrada. Após o processamento da FDCT, existem três possíveis caminhos de dados, que são utilizados de acordo com o tipo de amostra de entrada e o tipo de predição que foi utilizado para a geração desta amostra.

O primeiro caminho de dados é composto pelo *buffer* Y_{DC} e pela Hadamard 4x4. Este caminho de dados é usado somente se foi usada a predição intra-quadro 16x16 (RICHARDSON, 2003) para gerar a amostra de entrada. Neste caso, os elementos DC das matrizes 4x4 de entrada são processados pela Hadamard 4x4. O *buffer* é usado para armazenar, em grupos de 16 amostras (matrizes 4x4), os elementos DC das matrizes de entrada. Os resultados da Hadamard 4x4 direta são enviados para a quantização de componentes DC de luminância (Quant. Y_{DC} na Figura 4.10).

O segundo caminho de dados é sempre usado para todos os modos de predição. Este caminho é composto pelo *buffer* C_{DC} e pela Hadamard 2x2 e é usado para processar todas as amostras DC de crominância. O *buffer* C_{DC} armazena oito componentes DC de crominância (duas matrizes 2x2, uma de Cb e outra de Cr). Após o cálculo da Hadamard 2x2 direta, os resultados são enviados para a quantização de componentes DC de crominância (Quant. C_{DC} na Figura 4.10).

Finalmente, o último caminho de dados é usado para transferir diretamente os resultados da FDCT para a quantização (Quant. YC na Figura 4.10). Este caminho é usado para todos os resultados da FDCT, que não devem ser processados por nenhuma Hadamard.

O módulo T paralelo possui 16 entradas paralelas, com 8 bits por entrada e um conjunto de 16 entradas é consumido a cada ciclo de *clock*. Por outro lado, este módulo possui três saídas distintas, duas com 16 amostras paralelas e uma com 4 amostras paralelas e cada amostra de saída é composta por 15 bits.

4.3.3 Resultados de Síntese do Módulo T Paralelo

Todas as arquiteturas desenvolvidas para o módulo T paralelo foram descritas em VHDL, utilizando a ferramenta ISE da Xilinx. A síntese das arquiteturas foi direcionada ao FPGA VP70, da família Virtex II Pró da Xilinx (XILINX, 2005). Os principais resultados de síntese são apresentados na Tabela 4.6.

A Tabela 4.6 omite os resultados de síntese para os *buffers*. No entanto, o módulo T reúne todas as arquiteturas apresentadas na Tabela 4.6 e os *buffers* de sincronização de dados apresentados na Figura 4.10.

Tabela 4.6: Resultados de síntese do módulo T paralelo

	Elementos Lógicos	Freqüência (MHz)	Taxa de Processamento (Mamostras/s)
Hadamard 4x4 Direta	928	303,6	4.857,6
DCT 4x4 Direta	656	319,7	5.115,2
Hadamard 2x2 Direta	108	311,4	4.982,4
Controle	129	561,7	–
Módulo T Paralelo	1783	303,6	4.857,6

Dispositivo 2VP70FF1517-6

O principal dado apresentado na Tabela 4.6 é a taxa de processamento alcançada pelas arquiteturas. O módulo T paralelo é capaz de gerar dados a uma taxa superior 4,8 bilhões de amostras por segundo, sendo capaz de processar mais de 1.561 quadros HDTV por segundo. Essa taxa possibilita, com larga margem de folga, o processamento de vídeos HDTV em tempo real (30 quadros por segundo). Esta margem de folga acentuada torna esta solução extremamente flexível, sendo útil para diferentes arquiteturas de codificadores H.264/AVC. Mas o ponto principal diz respeito à redução do tempo de espera da predição intra-quadro no codificador, caso a solução paralela seja utilizada no codificador completo.

Uma comparação entre a implementação serial e paralela para o módulo das transformadas diretas está apresentada na Tabela 4.7. Da Tabela 4.7 pode-se notar que o módulo T paralelo, além de possuir uma taxa de processamento muito superior do que a taxa de processamento do módulo T serial (cerca de 35 vezes), também utiliza um número menor de elementos lógicos. Isto se deve, principalmente, em função da redução no tamanho do módulo de controle e, também, em função da redução drástica do número de *buffers* necessários para manter o sincronismo.

Tabela 4.7: Comparação entre os módulos T serial e paralelo

Módulo	Elementos Lógicos	Freqüência (MHz)	Taxa de Processamento (Mamostras/s)
T serial	2.359	138,7	138,7
T paralelo	1.783	303,6	4.857,6

Dispositivo 2VP70FF1517-6

Esta arquitetura foi publicada em (PORTO, 2007) e exceto pela arquitetura para o módulo T serial desenvolvida nesta tese e publicada em (AGOSTINI, 2006b), nenhuma outra arquitetura para o módulo T completo foi encontrada na literatura.

4.4 Arquitetura Serial para o Módulo das Transformadas Inversas

A primeira implementação do módulo das transformadas inversas (T^{-1}) foi construída usando transformadas seriais sem separabilidade, conforme foi discutido na seção 4.1 deste texto. Então, as arquiteturas da IDCT 2-D, da Hadamard 2-D 4x4 inversa e da Hadamard 2-D 2x2 inversa foram desenvolvidas de acordo com esta estratégia. Esta seção irá apresentar os algoritmos e as arquiteturas desenvolvidas para as transformadas inversas utilizando a estratégia serial sem separabilidade, além do projeto completo do módulo T^{-1}. O projeto do módulo T^{-1} serial, do mesmo modo que o projeto do módulo T serial, teve por objetivo principal a facilidade de integração deste módulo com os demais módulos do codificador H.264/AVC. Então, de maneira similar ao que foi desenvolvido no módulo T, foi construída uma arquitetura para o módulo T^{-1} que consome e produz uma amostra a cada ciclo de *clock*.

4.4.1 A Arquitetura da IDCT 2-D 4x4 Serial

O algoritmo utilizado para o cálculo da IDCT 2-D está apresentado na Tabela 4.8 e foi extraído da definição apresentada em (20). Na Tabela 4.8 $\mathbf{a_0}$ a $\mathbf{a_{15}}$, $\mathbf{b_0}$ a $\mathbf{b_{15}}$ e $\mathbf{c_0}$ a $\mathbf{c_{15}}$ são variáveis internas, que são úteis à implementação em *pipeline* desenvolvida. As saídas são representadas pelas variáveis $\mathbf{S_0}$ a $\mathbf{S_{15}}$ e as entradas são as variáveis $\mathbf{X_0}$ a $\mathbf{X_{15}}$.

Tabela 4.8: Algoritmo utilizado para o cálculo da IDCT 2-D

$a_0 = X_0 + X_8$	$b_0 = a_0 + a_1$	$c_0 = b_0 + b_2$	$S_0 = c_0 + c_1$
$a_1 = X_4 + X_{12}/2$	$b_1 = a_2 + a_3$	$c_1 = b_1 + b_3/2$	$S_1 = c_2 + c_3$
$a_2 = X_1 + X_9$	$b_2 = a_4 + a_5$	$c_2 = b_0 - b_2$	$S_2 = c_2 - c_3$
$a_3 = X_5 + X_{13}/2$	$b_3 = a_6 + a_7$	$c_3 = b_1/2 - b_3$	$S_3 = c_0 - c_1$
$a_4 = X_2 + X_{10}$	$b_4 = a_8 + a_9$	$c_4 = b_4 + b_6$	$S_4 = c_4 + c_5$
$a_5 = X_6 + X_{14}/2$	$b_5 = a_{10} + a_{11}$	$c_5 = b_5 + b_7/2$	$S_5 = c_6 + c_7$
$a_6 = X_3 + X_{11}$	$b_6 = a_{12} + a_{13}$	$c_6 = b_4 - b_6$	$S_6 = c_6 - c_7$
$a_7 = X_7 + X_{15}/2$	$b_7 = a_{14} + a_{15}$	$c_7 = b_5/2 - b_7$	$S_7 = c_4 - c_5$
$a_8 = X_0 - X_8$	$b_8 = a_8 - a_9$	$c_8 = b_8 + b_{10}$	$S_8 = c_8 + c_9$
$a_9 = X_4/2 - X_{12}$	$b_9 = a_{10} - a_{11}$	$c_9 = b_9 + b_{11}/2$	$S_9 = c_{10} + c_{11}$
$a_{10} = X_1 - X_9$	$b_{10} = a_{12} - a_{13}$	$c_{10} = b_8 - b_{10}$	$S_{10} = c_{10} - c_{11}$
$a_{11} = X_5/2 - X_{13}$	$b_{11} = a_{14} - a_{15}$	$c_{11} = b_9/2 - b_{11}$	$S_{11} = c_8 - c_9$
$a_{12} = X_2 - X_{10}$	$b_{12} = a_0 - a_1$	$c_{12} = b_{12} + b_{14}$	$S_{12} = c_{12} + c_{13}$
$a_{13} = X_6/2 - X_{14}$	$b_{13} = a_2 - a_3$	$c_{13} = b_{13} + b_{15}/2$	$S_{13} = c_{14} + c_{15}$
$a_{14} = X_3 - X_{11}$	$b_{14} = a_4 - a_5$	$c_{14} = b_{12} - b_{14}$	$S_{14} = c_{14} - c_{15}$
$a_{15} = X_7/2 - X_{15}$	$b_{15} = a_6 - a_7$	$c_{15} = b_{13}/2 - b_{15}$	$S_{15} = c_{12} - c_{13}$

A arquitetura desenvolvida para implementar a IDCT 2-D é muito parecida com a arquitetura para cálculo da FDCT 2-D, apresentada na Figura 4.6. A principal diferença está nos deslocamentos que, ao invés de ser para a esquerda como da FDCT 2-D (multiplicações) são para a direita (divisões). A arquitetura da IDCT 2-D sem separabilidade foi projetada em um *pipeline* de quatro estágios, sendo que cada estágio utiliza 16 ciclos de *clock* para finalizar as suas operações. Novamente, apenas um operador é utilizado por estágio de *pipeline*. A latência da arquitetura sem separabilidade é de 64 ciclos de *clock* e são utilizados 39 bits de entrada e 43 bits de saída.

4.4.2 A Arquitetura da Hadamard 2-D 4x4 Inversa Serial

O algoritmo para o cálculo da Hadamard 4x4 inversa está apresentado na Tabela 4.9. Este algoritmo segue a definição de (21). Na Tabela 4.9, $\mathbf{a_0}$ a $\mathbf{a_{15}}$, $\mathbf{b_0}$ a $\mathbf{b_{15}}$ e $\mathbf{c_0}$ a $\mathbf{c_{15}}$ são variáveis internas, que são úteis para o *pipeline* desenvolvido. As saídas são representadas pelas variáveis $\mathbf{S_0}$ a $\mathbf{S_{15}}$ e as entradas são as variáveis $\mathbf{W_0}$ a $\mathbf{W_{15}}$.

A arquitetura desenvolvida para implementar a Hadamard 2-D 4x4 inversa é quase idêntica à arquitetura para cálculo da Hadamard 2-D 4x4 direta, apresentada na Figura 4.2. A única diferença está nos deslocamentos na saída que estão presentes na arquitetura direta e que não existem na arquitetura inversa.

A arquitetura da Hadamard 2-D 4x4 inversa serial foi projetada em um *pipeline* de quatro estágios com 16 ciclos de *clock* sendo usados em cada estágio. Novamente apenas um operador é utilizado por estágio de *pipeline*. A latência da arquitetura sem

separabilidade é de 64 ciclos de *clock* e as entradas possuem 14 bits, enquanto que as saídas possuem 18 bits.

Tabela 4.9: Algoritmo utilizado para o cálculo da Hadamard 2-D 4x4 inversa

$a_0 = W_0 + W_4$	$B_0 = a_0 + a_1$	$c_0 = b_0 + b_1$	$S_0 = c_0 + c_1$
$a_1 = W_8 + W_{12}$	$B_1 = a_2 + a_3$	$c_1 = b_2 + b_3$	$S_1 = c_0 - c_1$
$a_2 = W_1 + W_5$	$B_2 = a_4 + a_5$	$c_2 = b_0 - b_1$	$S_2 = c_2 - c_3$
$a_3 = W_9 + W_{13}$	$B_3 = a_6 + a_7$	$c_3 = b_2 - b_3$	$S_3 = c_2 + c_3$
$a_4 = W_2 + W_6$	$B_4 = a_0 - a_1$	$c_4 = b_4 + b_5$	$S_4 = c_4 + c_5$
$a_5 = W_{10} + W_{14}$	$B_5 = a_2 - a_3$	$c_5 = b_6 + b_7$	$S_5 = c_4 - c_5$
$a_6 = W_3 + W_7$	$B_6 = a_4 - a_5$	$c_6 = b_4 - b_5$	$S_6 = c_6 - c_7$
$a_7 = W_{11} + W_{15}$	$B_7 = a_6 - a_7$	$c_7 = b_6 - b_7$	$S_7 = c_6 + c_7$
$a_8 = W_0 - W_4$	$B_8 = a_8 - a_9$	$c_8 = b_8 + b_9$	$S_8 = c_8 + c_9$
$a_9 = W_8 - W_{12}$	$B_9 = a_{10} - a_{11}$	$c_9 = b_{10} + b_{11}$	$S_9 = c_8 - c_9$
$a_{10} = W_1 - W_5$	$B_{10} = a_{12} - a_{13}$	$c_{10} = b_8 - b_9$	$S_{10} = c_{10} - c_{11}$
$a_{11} = W_9 - W_{13}$	$B_{11} = a_{14} - a_{15}$	$c_{11} = b_{10} - b_{11}$	$S_{11} = c_{10} + c_{11}$
$a_{12} = W_2 - W_6$	$B_{12} = a_8 + a_9$	$c_{12} = b_{12} + b_{13}$	$S_{12} = c_{12} + c_{13}$
$a_{13} = W_{10} - W_{14}$	$B_{13} = a_{10} + a_{11}$	$c_{13} = b_{14} + b_{15}$	$S_{13} = c_{12} - c_{13}$
$a_{14} = W_3 - W_7$	$B_{14} = a_{12} + a_{13}$	$c_{14} = b_{12} - b_{13}$	$S_{14} = c_{14} - c_{15}$
$a_{15} = W_{11} - W_{15}$	$B_{15} = a_{14} + a_{15}$	$c_{15} = b_{14} - b_{15}$	$S_{15} = c_{14} + c_{15}$

4.4.3 A Arquitetura da Hadamard 2-D 2x2 Inversa Serial

Como já foi mencionado na seção 4.2.2, o algoritmo para o cálculo da Hadamard 2x2 direta ou inversa é o idêntico e foi apresentado na Tabela 4.4. Em função disso, a arquitetura desenvolvida para a Hadamard 2x2 inversa serial é a mesma desenvolvida para a versão direta desta transformada e que foi apresentada na seção 4.2.2. Esta arquitetura foi desenvolvida em um *pipeline* de dois estágios, um operador é utilizado por estágio e a latência da arquitetura é de 8 ciclos de *clock*.

4.4.4 Arquitetura do Módulo T^{-1} Serial

O projeto da arquitetura da primeira implementação do módulo T^{-1}, como na arquitetura do módulo T, teve por objetivo gerar uma solução arquitetural capaz de sincronizar as operações internas ao módulo T^{-1}, fazendo com que o módulo T^{-1} consuma sempre uma mesma quantidade de ciclos de *clock*, não importando o modo de predição utilizado. Por isso, novamente uma FIFO foi inserida antes da IDCT 2-D, para atrasar a sua entrada quando o modo é não intra-quadro 16x16 e, deste modo, sincronizar as operações. Este sincronismo facilita a integração da arquitetura aos demais módulos do compressor, pois, após o *pipeline* estar cheio, um valor válido é gerado pelo módulo T^{-1} a cada ciclo de *clock*, não importando o modo de previsão que fora adotado. A arquitetura do módulo T^{-1} está apresentada na Figura 4.11.

Além da FIFO no caminho da IDCT 2-D, na Figura 4.11, outros *buffers* foram necessários para garantir o sincronismo. Foi usado um *buffer* para armazenar os componentes DC de crominância (*buffer* C_{DC} na Figura 4.11). Outros três *buffers* (Y_{AC}, Cb_{AC} e Cr_{AC}) foram usados para armazenar os valores AC de luminância e crominância enquanto as Hadamards inversas realizam seus cálculos, garantindo, deste modo, o sincronismo entre as operações.

Na figura 4.11 é importante ressaltar a presença do módulo da quantização inversa (destacado na Figura 4.11) entre as operações do módulo T^{-1}. Como já foi explicado no capítulo 3, a operação do módulo T^{-1} é dividida em duas etapas, uma antes da quantização inversa e outra após a quantização inversa.

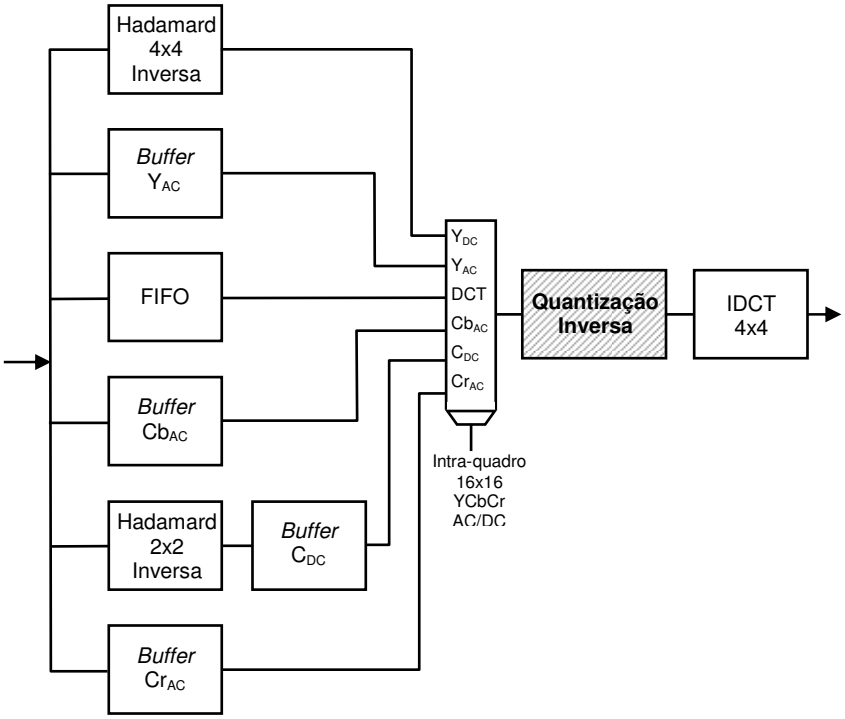

Figura 4.11: Diagrama em blocos da arquitetura do módulo T^{-1} serial

Todos os caminhos de dados que entram no módulo T^{-1} possuem a mesma latência, que é de 133 ciclos de *clock* e, novamente, a geração de um valor por ciclo de *clock* na saída pode ser garantida se a taxa de entrada for também de um valor por ciclo. Os *buffers* e a FIFO foram desenvolvidos utilizando registradores, mas poderiam ter sido implementados com memória. A opção pelo uso de registradores foi tomada com o intuito de minimizar o uso de memória RAM, disponibilizando mais memória para os outros módulos do codec H.264/AVC, como a estimação de movimento, a compensação de movimento, a predição intra-quadro e o filtro redutor do efeito de bloco, que possuem no uso de memória um requisito extremamente crítico.

Na saída do módulo T^{-1} os seis bits menos significativos são desconsiderados, de acordo com o que especifica o padrão H.264/AVC. Esta operação não está apresentada na Figura 4.11.

O módulo T^{-1} utiliza 14 bits para os coeficientes quantizados de sua entrada e gera 37 bits (já considerando o seis bits desconsiderados na saída) para cada amostra de resíduo reconstruído na sua saída. O número de bits de saída da transformada inversa deveria ser o mesmo número de bits da entrada da transformada direta, ou seja, 8 bits. Após uma análise inicial das operações aritméticas realizadas, não é possível garantir que as operações possam ter seus resultados finais acumulados representados com este número de bits. Mas as operações realizadas sobre os dados reconstruídos devem utilizar o mesmo número de bits por amostras utilizados no quadro original. Por isso, uma análise mais criteriosa sobre a faixa dinâmica dos resultados se faz necessária, para determinar, com precisão, quais bits podem ser desprezados e em que momento dos cálculos estes bits podem ser desconsiderados. A primeira versão implementada da

arquitetura do módulo T^{-1} não resolve este problema e gera, em sua saída, um número muito maior de bits do que o esperado.

A parte de controle do módulo T^{-1} é responsável pela sincronização entre as operações dos blocos internos do módulo. O controle foi projetado através de máquinas de estados finitos que interagem para dividir a tarefa de gerar os sinais de controle, a partir dos sinais externos e do dado que está sendo processado.

4.4.5 Resultados de Síntese do Módulo T^{-1} Serial

A Tabela 4.10 apresenta os resultados de síntese dos principais componentes do módulo T^{-1}. Também estão apresentados os resultados do módulo T^{-1} completo, incluindo os resultados do módulo da quantização inversa. Como a quantização inversa está inserida dentro do módulo T^{-1}, não é possível sintetizar o módulo T^{-1} sem considerar o módulo da quantização inversa. Novamente, o dispositivo alvo foi um FPGA Virtex-II Pro VP70 (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada foi o Synplify Pro da Synplicity (SYNPLICITY, 2007).

Tabela 4.10: Resultados de síntese do módulo T^{-1} serial

Bloco	Elementos Lógicos	Freqüência (MHz)	Taxa de Processamento (Mamostras/s)
Controle	1008	166,9	166,9
IDCT 2-D 4x4	1682	132,1	132,1
Hadamard 2-D 4x4 Inversa	605	176,8	176,8
Hadamard 2-D 2x2 Inversa	98	201,7	201,7
FIFO	76	320,6	320,6
***Buffer* YDC**	39	318,1	318,1
***Buffer* YAC**	181	243,7	243,7
***Buffer* CDC**	35	318,1	318,1
***Buffer* CbAC**	76	320,6	320,6
***Buffer* CrAC**	76	320,6	320,6
Quantização Inversa	284	226,7	226,7
Bloco T^{-1}	3856	121,4	121,4

Dispositivo 2VP70FF1517-6

A taxa de processamento dos módulos apresentados na Tabela 4.10 indica que todos os módulos estão aptos a serem utilizados em um codec H.264/AVC para HDTV. Nos principais módulos formadores do módulo T^{-1} existe uma folga mínima superior a 38 milhões de amostras por segundo em relação à exigência de processamento de um codec HDTV.

A maior taxa de processamento entre as transformadas é atingida pela arquitetura mais simples, ou seja, pela arquitetura da Hadamard 2-D 2x2. Nesta arquitetura a taxa de processamento chega a 201 milhões de amostras por segundo. O menor consumo de

recursos entre as transformadas também é desta arquitetura, que utiliza apenas 98 elementos lógicos.

A IDCT 4x4 foi o módulo com a mais baixa freqüência de operação, atingindo 132,1 MHz, com uma taxa de processamento superior a 132 milhões de amostras por segundo.

O módulo da IDCT 4x4 foi o maior responsável pelo consumo de elementos lógicos do FPGA, seguido pelo módulo de controle e pelo módulo da transformada Hadamard 4x4 inversa.

A arquitetura da IDCT 2-D utiliza mais elementos lógicos do que a arquitetura da Hadamard 2-D 4x4 inversa porque a largura de bits de entrada da IDCT é mais elevada do que a largura de bits da Hadamard 2-D 4x4 inversa.

O módulo da quantização inversa (Q^{-1}) atingiu 226,7 MHz em sua síntese, indicando que este módulo não é o gargalo de desempenho do módulo T^{-1}. Além disso, este módulo utilizou apenas 284 elementos lógicos.

O módulo T^{-1} (incluindo a quantização inversa), por sua vez, também atingiu os requisitos de HDTV, uma vez que atingiu 121,4MHz, permitindo uma taxa de processamento superior a 121 milhões de amostras por segundo. Deste modo, a implementação do módulo T^{-1} junto com a implementação do módulo Q^{-1} possui uma folga mínima de desempenho superior a 28 milhões de amostras por segundo se o projeto for direcionado para HDTV. Assim, devido à elevada resolução definida para HDTV, qualquer codec H.264/AVC direcionado para os padrões de menor resolução poderá utilizar a arquitetura dos módulos T^{-1} e Q^{-1} desenvolvidos e apresentados neste trabalho. Do ponto de vista do consumo de elementos lógicos do FPGA alvo, a implementação dos módulos T^{-1} e Q^{-1} utilizou cerca de 11% do total de elementos lógicos disponíveis, em um total de 3.856 elementos lógicos utilizados.

Os resultados obtidos com o desenvolvimento do módulo T^{-1} foram publicadas em (AGOSTINI, 2006c). Nenhuma implementação em hardware para o módulo T^{-1} completo foi encontrada na literatura. Deste modo, não foi possível estabelecer comparações do módulo desenvolvido neste trabalho com outras soluções.

4.5 Arquitetura Multitransformada com Paralelismo Programável

O desenvolvimento das arquiteturas de transformadas diretas e inversas sem separabilidade, apresentadas nas seções 4.2 e 4.4, conduziu a constatação que existem interessantes similaridades entre os algoritmos e arquiteturas destas transformadas. Deste modo, surgiu a idéia de construir uma arquitetura capaz de realizar o cálculo de qualquer transformada dos módulos T e T^{-1}, isto é, a FDCT 4x4, a IDCT 4x4, a Hadamard Direta 4x4 a Hadamard Inversa 4x4 e a Hadamard 2x2. Esta arquitetura, por realizar o cálculo de todas as transformadas, foi batizada de arquitetura multitransformada. Além desta idéia, em função da exploração do espaço de projeto das transformadas que foi apresentado na seção 4.1, surgiu uma idéia ainda mais interessante, qual seja: desenvolver a arquitetura multitransformada com paralelismo programável, isto é, o nível de paralelismo utilizado nos cálculos das transformadas seria definido pelo usuário da arquitetura. Então estas idéias foram exploradas e a arquitetura foi desenvolvida e está apresentada nesta seção do texto.

A primeira etapa no desenvolvimento desta arquitetura foi avaliar criteriosamente as similaridades dos algoritmos apresentados nas Tabelas 4.1, 4.3, 4.4, 4.8 e 4.9. Após esta

análise, foi constatado que a arquitetura multitransformada realmente seria viável. Uma importante similaridade entre estes algoritmos é que, para um mesmo passo, existe sempre o mesmo número de adições e subtrações para todas as transformadas 4x4.

Em função do que foi desenvolvido nas seções 4.1, 4.2, 4.3 e 4.4, foi possível constatar que a implementação desta idéia apresentava dois desafios principais: (a) desenvolver um gerenciamento de entrada e saída programável capaz de suportar diferentes níveis de paralelismo e (b) desenvolver uma arquitetura multitransformada com alto desempenho, capaz de atingir a taxa de processamento exigida pelos diferentes níveis de paralelismo. As soluções para estes dois desafios estão muito relacionadas e serão detalhadas nas próximas seções.

Os níveis de paralelismo definidos para as entradas são de 1, 2, 4, 8 ou 16 amostras por ciclo. Isso implica que a matriz 4x4 de entrada pode ser lida amostra por amostra, de uma maneira completamente serial, ou, no outro extremo, a matriz completa pode ser lida em um único ciclo. A arquitetura multitransformada deve ser capaz de suportar todos estes níveis de paralelismo. O caso mais complicado de resolver para a multitransformada é justamente o caso do maior nível de paralelismo. Se 16 amostras são lidas a cada ciclo, a arquitetura deve ser capaz de processar todas as 16 amostras em um único ciclo para que, no próximo ciclo, outras 16 amostras possam ser processadas. Por isso, optou-se por usar a implementação paralela com pipeline apresentada na seção 4.1.4 como referência para desenvolver o núcleo da arquitetura multitransformada, pois esta solução suporta o processamento de até 16 amostras por ciclo. Assim, o gerenciamento de entrada e saída deverá agrupar as amostras de entrada em grupos de 16 amostras e, então, prover estas amostras para a arquitetura multitransformada. Por outro lado, o gerenciamento também é responsável por receber os 16 coeficientes gerados pela arquitetura e disponibilizá-los na saída na taxa definida pelo nível de paralelismo empregado.

Para flexibilizar ainda mais o uso desta arquitetura, optou-se por desenvolver todos os módulos com o número de bits de entrada parametrizável. Deste modo, a arquitetura pode ser utilizada em implementações que exijam faixas dinâmicas diferenciadas.

As duas próximas seções apresentam em mais detalhes estas duas partes principais da arquitetura, enquanto que a seção 4.5.3 apresenta os resultados de síntese.

4.5.1 Gerenciamento de Entrada e Saída Programável

A solução desenvolvida para gerenciar o nível de paralelismo da entrada e da saída da arquitetura multitransformada considera que o usuário irá indicar o nível de paralelismo desejado através de um sinal externo, chamado de PAR neste trabalho. Os níveis de paralelismo possíveis são 1, 2, 4, 8 e 16, como já foi mencionado. Estes níveis foram escolhidos porque somente eles permitem processar a matriz 4x4 de entrada com **n** ciclos de *clock*, sendo **n** sempre um número inteiro. Se o paralelismo for 1, então **n** será 16; se o paralelismo for 2, então **n** será 8, e assim por diante, até que o nível de paralelismo seja 16, onde **n** será igual a 1.

O sinal PAR foi codificado para representar os cinco possíveis níveis de paralelismo. Deste modo, a parte de controle do gerenciamento de entrada e saída deverá decodificar este sinal para indicar para a parte operativa quais as operações devem ser realizadas. A Tabela 4.11 apresenta os códigos do sinal PAR. O sinal PAR é utilizado em todas as operações do gerenciador de paralelismo, controlando os multiplexadores, gerando os sinais de habilitação de escrita nos registradores internos, etc.

Tabela 4.11: Códigos do sinal PAR indicando o nível de paralelismo

Nível de Paralelismo	Sinal PAR	
	Decimal	Binário
1 amostra por ciclo	0	000
2 amostras por ciclo	1	001
4 amostras por ciclo	2	010
8 amostras por ciclo	3	011
16 amostras por ciclo	4	100

A Figura 4.12 apresenta a arquitetura do gerenciador de paralelismo de entrada. A barreira de registradores apresentada na Figura 4.12 (R_0 a R_{14}) é utilizada para os níveis de paralelismo 1, 2, 4 e 8. Os registradores recebem como entrada de uma a oito diferentes entradas externas, dependendo do nível de paralelismo selecionado. Os registradores apresentados na Figura 4.12 podem ter suas entradas conectadas diretamente em uma das entradas externas (IN_0, IN_2, IN_4, IN_6, IN_8, IN_{10}, IN_{12} ou IN_{14}) ou no registrador imediatamente anterior. As saídas estão conectadas no próximo registrador e no multiplexador que seleciona a entrada de cada operador.

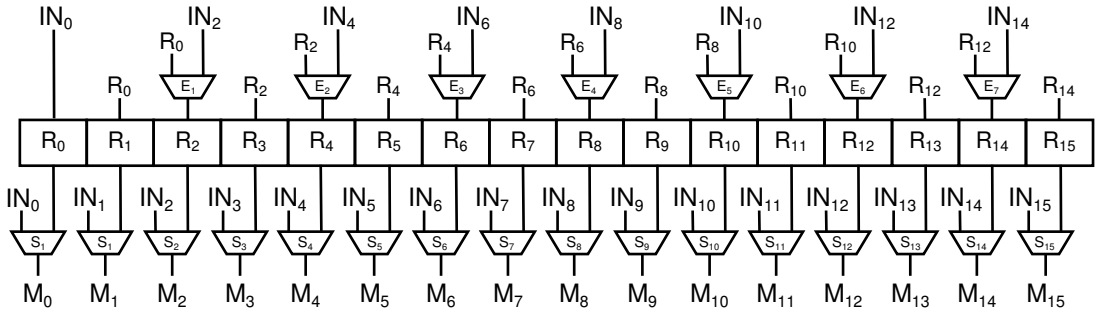

Figura 4.12: Arquitetura do gerenciador de paralelismo de entrada da arquitetura multitransformada com multiparalelismo

Na Figura 4.12, os sinais de controle dos multiplexadores foram omitidos, para permitir uma maior clareza. Quando o nível de paralelismo selecionado é 16, então, com o objetivo de maximizar o desempenho, as 16 entradas paralelas (IN_0 a IN_{15}) não passam pela barreira de registradores e são entregues diretamente para a arquitetura da multitransformada (M_0 a M_{15}), passando primeiro pelo multiplexador de saída. Este conjunto de multiplexadores é controlado pelo mesmo sinal e entrega para a arquitetura da multitransformada os 16 valores paralelos de entrada.

O controle dos multiplexadores que selecionam a entrada dos registradores é um pouco mais complicado, pois a escrita nos registradores depende do nível de paralelismo selecionado. A Tabela 4.12 apresenta as entradas dos registradores de acordo com o nível de paralelismo selecionado. Na Tabela 4.12 estão destacados, em cinza, os pontos onde as entradas externas são conectadas aos registradores.

A partir do que está apresentado na Tabela 4.12, é possível gerar os sinais de controle para os multiplexadores de entrada.

Tabela 4.12: Entradas dos registradores R0 a R14 de acordo com o nível de paralelismo

Nível de Paralelismo			
1 amostra/ciclo	2 amostras/ciclo	4 amostras/ciclo	8 amostras/ciclo
$R_0 \leftarrow IN_0$	$R_0 \leftarrow IN_0$	$R_0 \leftarrow IN_0$	$R_0 \leftarrow IN_0$
$R_1 \leftarrow R_0$	$R_1 \leftarrow R_0$	$R_1 \leftarrow R_0$	$R_1 \leftarrow R_0$
$R_2 \leftarrow R_1$	$R_2 \leftarrow R_1$	$R_2 \leftarrow R_1$	$R_1 \leftarrow IN_2$
$R_3 \leftarrow R_2$	$R_3 \leftarrow R_2$	$R_3 \leftarrow R_2$	$R_3 \leftarrow R_2$
$R_4 \leftarrow R_3$	$R_4 \leftarrow R_3$	$R_4 \leftarrow IN_4$	$R_3 \leftarrow IN_4$
$R_5 \leftarrow R_4$	$R_5 \leftarrow R_4$	$R_5 \leftarrow R_4$	$R_5 \leftarrow R_4$
$R_6 \leftarrow R_5$	$R_6 \leftarrow R_5$	$R_6 \leftarrow R_5$	$R_5 \leftarrow IN_6$
$R_7 \leftarrow R_6$	$R_7 \leftarrow R_6$	$R_7 \leftarrow R_6$	$R_7 \leftarrow R_6$
$R_8 \leftarrow R_7$	$R_8 \leftarrow IN_8$	$R_8 \leftarrow IN_8$	$R_7 \leftarrow IN_8$
$R_9 \leftarrow R_8$	$R_9 \leftarrow R_8$	$R_9 \leftarrow R_8$	$R_9 \leftarrow R_8$
$R_{10} \leftarrow R_9$	$R_{10} \leftarrow R_9$	$R_{10} \leftarrow R_9$	$R_9 \leftarrow IN_{10}$
$R_{11} \leftarrow R_{10}$	$R_{11} \leftarrow R_{10}$	$R_{11} \leftarrow R_{10}$	$R_{11} \leftarrow R_{10}$
$R_{12} \leftarrow R_{11}$	$R_{12} \leftarrow R_{11}$	$R_{12} \leftarrow IN_{12}$	$R_{11} \leftarrow IN_{12}$
$R_{13} \leftarrow R_{12}$	$R_{13} \leftarrow R_{12}$	$R_{13} \leftarrow R_{12}$	$R_{13} \leftarrow R_{12}$
$R_{14} \leftarrow R_{13}$	$R_{14} \leftarrow R_{13}$	$R_{14} \leftarrow R_{12}$	$R_{13} \leftarrow IN_{14}$
$R_{15} \leftarrow R_{14}$	$R_{15} \leftarrow R_{14}$	$R_{15} \leftarrow R_{14}$	$R_{15} \leftarrow R_{14}$

É importante destacar que o número de entradas utilizadas depende diretamente do nível de paralelismo selecionado. A Tabela 4.13 apresenta esta relação indicando quais são as entradas utilizadas para cada nível de paralelismo.

Tabela 4.13: Entradas utilizadas para cada nível de paralelismo

Nível de Paralelismo	Entradas Utilizadas
1	IN_0
2	IN_0 e IN_8
4	IN_0, IN_4, IN_8 e IN_{12}
8	IN_0, IN_2, IN_4, IN_6, IN_8, IN_{10}, IN_{12} e IN_{14}
16	IN_0, IN_1, IN_2, IN_3, IN_4, IN_5, IN_6, IN_7, IN_8, IN_9, IN_{10}, IN_{11}, IN_{12}, IN_{13}, IN_{14} e IN_{15}

A implementação do gerenciador do paralelismo da saída possui algumas semelhanças com a implementação do gerenciador de entrada.

Com um paralelismo de nível 16, o gerenciador de saída entrega diretamente os 16 valores gerados pela arquitetura multitransformada para a saída, sem passar pelos registradores. Esta operação é realizada para maximizar o desempenho quando o paralelismo é de 16 amostras por ciclo. Nos outros casos, os resultados da arquitetura

multitransformada são armazenados nos registradores e entregues na ordem correta nas saídas, a partir de deslocamentos sobre os dados armazenados. A Figura 4.13 apresenta a arquitetura do gerenciador de paralelismo de saída onde, novamente, os sinais de controle foram omitidos.

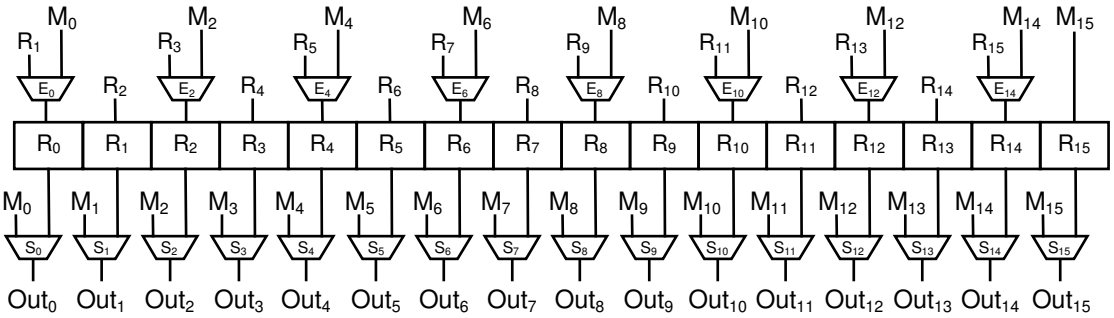

Figura 4.13: Arquitetura do gerenciador de paralelismo de saída da arquitetura multitransformada com multiparalelismo

Do mesmo modo que ocorre no gerenciador de paralelismo de entrada, o número de saídas utilizadas no gerenciador de paralelismo de saída depende do nível de paralelismo. A Tabela 4.14 apresenta esta relação.

Tabela 4.14: Saídas utilizadas para cada nível de paralelismo

Nível de Paralelismo	Saídas Utilizadas
1	Out_0
2	Out_0 e Out_8
4	Out_0, Out_4, Out_8 e Out_{12}
8	Out_0, Out_2, Out_4, Out_6, Out_8, Out_{10}, Out_{12} e Out_{14}
16	Out_0, Out_1, Out_2, Out_3, Out_4, Out_5, Out_6, Out_7, Out_8, Out_9, Out_{10}, Out_{11}, Out_{12}, Out_{13}, Out_{14} e Out_{15}

4.5.2 Arquitetura Multitransformada de Alto Desempenho

A solução adotada para desenvolver o núcleo da arquitetura multitransformada foi muito similar a que foi utilizada na arquitetura paralela com pipeline apresentada na seção 4.1.4 do texto. A principal diferença reside nos algoritmos implementados, pois para minimizar o consumo de recursos e a velocidade de processamento, as operações internas dos algoritmos apresentados nas Tabelas 4.1, 4.3, 4.4, 4.8 e 4.9 foram reorganizadas. Assim, foi possível agrupar as operações similares para todas as transformadas e compartilhar os operadores de maneira eficiente entre elas.

A Tabela 4.15 apresenta os novos algoritmos com operações reagrupadas. Esta tabela apresenta as operações de todas as transformadas implementadas na arquitetura multitransformada. Os algoritmos apresentados na Tabela 4.15 apresentam as operações agrupadas por estágio de pipeline para cada uma das transformadas.

Tabela 4.15: Algoritmos com operações reagrupadas para a multitransformada

	Hadamard 4x4 Direta	FDCT	IDCT	Hadamard 4x4 Inversa	Hadamard 2x2
ESTÁGIO A	$a_0 = X_0 + X_{12}$	$a_0 = X_0 + X_{12}$	$a_0 = X_0 + X_8$	$a_0 = X_0 + X_{12}$	$a_0 = X_0 + X_2$
	$a_1 = X_4 + X_8$	$a_1 = X_4 + X_8$	$a_1 = X_4 + X_{12}/2$	$a_1 = X_4 + X_8$	*don't care*
	$a_2 = X_1 + X_{13}$	$a_2 = X_1 + X_{13}$	$a_2 = X_1 + X_9$	$a_2 = X_1 + X_{13}$	*don't care*
	$a_3 = X_5 + X_9$	$a_3 = X_5 + X_9$	$a_3 = X_5 + X_{13}/2$	$a_3 = X_5 + X_9$	*don't care*
	$a_4 = X_2 + X_{14}$	$a_4 = X_2 + X_{14}$	$a_4 = X_2 + X_{10}$	$a_4 = X_2 + X_{14}$	*don't care*
	$a_5 = X_6 + X_{10}$	$a_5 = X_6 + X_{10}$	$a_5 = X_6 + X_{14}/2$	$a_5 = X_6 + X_{10}$	*don't care*
	$a_6 = X_3 + X_{15}$	$a_6 = X_3 + X_{15}$	$a_6 = X_3 + X_{11}$	$a_6 = X_3 + X_{15}$	$a_6 = X_3 + X_1$
	$a_7 = X_7 + X_{11}$	$a_7 = X_7 + X_{11}$	$a_7 = X_7 + X_{15}/2$	$a_7 = X_7 + X_{11}$	*don't care*
	$a_8 = X_0 - X_{12}$	$a_8 = X_0 - X_{12}$	$a_8 = X_0 - X_8$	$a_8 = X_0 - X_{12}$	$a_8 = X_0 - X_2$
	$a_9 = X_4 - X_8$	$a_9 = X_4 - X_8$	$a_9 = X_4/2 - X_{12}$	$a_9 = X_4 - X_8$	*don't care*
	$a_{10} = X_1 - X_{13}$	$a_{10} = X_1 - X_{13}$	$a_{10} = X_1 - X_9$	$a_{10} = X_1 - X_{13}$	*don't care*
	$a_{11} = X_5 - X_9$	$a_{11} = X_5 - X_9$	$a_{11} = X_5/2 - X_{13}$	$a_{11} = X_5 - X_9$	*don't care*
	$a_{12} = X_2 - X_{14}$	$a_{12} = X_2 - X_{14}$	$a_{12} = X_2 - X_{10}$	$a_{12} = X_2 - X_{14}$	*don't care*
	$a_{13} = X_6 - X_{10}$	$a_{13} = X_6 - X_{10}$	$a_{13} = X_6/2 - X_{14}$	$a_{13} = X_6 - X_{10}$	*don't care*
	$a_{14} = X_3 - X_{15}$	$a_{14} = X_3 - X_{15}$	$a_{14} = X_3 - X_{11}$	$a_{14} = X_3 - X_{15}$	$a_{14} = X_3 - X_1$
	$a_{15} = X_7 - X_{11}$	$a_{15} = X_7 - X_{11}$	$a_{15} = X_7/2 - X_{15}$	$a_{15} = X_7 - X_{11}$	*don't care*
ESTÁGIO B	$b_0 = a_0 + a_1$	$b_0 = a_0 + a_1$	$b_0 = a_0 + a_1$	$b_0 = a_0 + a_1$	$b_0 = a_0 + 0$
	$b_1 = a_2 + a_3$	$b_1 = a_2 + a_3$	$b_1 = a_2 + a_3$	$b_1 = a_2 + a_3$	*don't care*
	$b_2 = a_4 + a_5$	$b_2 = a_4 + a_5$	$b_2 = a_4 + a_5$	$b_2 = a_4 + a_5$	*don't care*
	$b_3 = a_6 + a_7$	$b_3 = a_6 + a_7$	$b_3 = a_6 + a_7$	$b_3 = a_6 + a_7$	$b_3 = a_6 + 0$
	$b_4 = a_8 + a_9$	$b_4 = 2*a_8 + a_9$	$b_4 = a_8 + a_9$	$b_4 = a_8 + a_9$	$b_4 = a_8 + 0$
	$b_5 = a_{10} + a_{11}$	$b_5 = 2*a_{10} + a_{11}$	$b_5 = a_{10} + a_{11}$	$b_5 = a_{10} + a_{11}$	*don't care*
	$b_6 = a_{12} + a_{13}$	$b_6 = 2*a_{12} + a_{13}$	$b_6 = a_{12} + a_{13}$	$b_6 = a_{12} + a_{13}$	*don't care*
	$b_7 = a_{14} + a_{15}$	$b_7 = 2*a_{14} + a_{15}$	$b_7 = a_{14} + a_{15}$	$b_7 = a_{14} + a_{15}$	$b_7 = a_{14} + 0$
	$b_8 = a_0 - a_1$	$b_8 = a_0 - a_1$	$b_8 = a_0 - a_1$	$b_8 = a_0 - a_1$	*don't care*
	$b_9 = a_2 - a_3$	$b_9 = a_2 - a_3$	$b_9 = a_2 - a_3$	$b_9 = a_2 - a_3$	*don't care*
	$b_{10} = a_4 - a_5$	$b_{10} = a_4 - a_5$	$b_{10} = a_4 - a_5$	$b_{10} = a_4 - a_5$	*don't care*
	$b_{11} = a_6 - a_7$	$b_{11} = a_6 - a_7$	$b_{11} = a_6 - a_7$	$b_{11} = a_6 - a_7$	*don't care*
	$b_{12} = a_8 - a_9$	$b_{12} = a_8 - 2*a_9$	$b_{12} = a_8 - a_9$	$b_{12} = a_8 - a_9$	*don't care*
	$b_{13} = a_{10} - a_{11}$	$b_{13} = a_{10} - 2*a_{11}$	$b_{13} = a_{10} - a_{11}$	$b_{13} = a_{10} - a_{11}$	*don't care*
	$b_{14} = a_{12} - a_{13}$	$b_{14} = a_{12} - 2*a_{13}$	$b_{14} = a_{12} - a_{13}$	$b_{14} = a_{12} - a_{13}$	*don't care*
	$b_{15} = a_{14} - a_{15}$	$b_{15} = a_{14} - 2*a_{15}$	$b_{15} = a_{14} - a_{15}$	$b_{15} = a_{14} - a_{15}$	*don't care*
ESTÁGIO C	$c_0 = b_0 + b_3$	$c_0 = b_0 + b_3$	$c_0 = b_0 + b_2$	$c_0 = b_0 + b_3$	$c_0 = b_0 + 0$
	$c_1 = b_1 + b_2$	$c_1 = b_1 + b_2$	$c_1 = b_1 + b_3/2$	$c_1 = b_1 + b_2$	$c_1 = 0 + b_3$
	$c_2 = b_0 - b_3$	$c_2 = b_0 - b_3$	$c_2 = b_0 - b_2$	$c_2 = b_0 - b_3$	*don't care*
	$c_3 = b_1 - b_2$	$c_3 = b_1 - b_2$	$c_3 = b_1/2 - b_3$	$c_3 = b_1 - b_2$	*don't care*
	$c_4 = b_4 + b_7$	$c_4 = b_4 + b_7$	$c_4 = b_4 + b_6$	$c_4 = b_4 + b_7$	$c_4 = b_4 + 0$
	$c_5 = b_5 + b_6$	$c_5 = b_5 + b_6$	$c_5 = b_5 + b_7/2$	$c_5 = b_5 + b_6$	$c_5 = 0 + b_7$
	$c_6 = b_4 - b_7$	$c_6 = b_4 - b_7$	$c_6 = b_4 - b_6$	$c_6 = b_4 - b_7$	*don't care*
	$c_7 = b_5 - b_6$	$c_7 = b_5 - b_6$	$c_7 = b_5/2 - b_7$	$c_7 = b_5 - b_6$	*don't care*
	$c_8 = b_8 + b_{11}$	$c_8 = b_8 + b_{11}$	$c_8 = b_{12} + b_{14}$	$c_8 = b_8 + b_{11}$	*don't care*
	$c_9 = b_9 + b_{10}$	$c_9 = b_9 + b_{10}$	$c_9 = b_{13} + b_{15}/2$	$c_9 = b_9 + b_{10}$	*don't care*
	$c_{10} = b_8 - b_{11}$	$c_{10} = b_8 - b_{11}$	$c_{10} = b_{12} - b_{14}$	$c_{10} = b_8 - b_{11}$	*don't care*
	$c_{11} = b_9 - b_{10}$	$c_{11} = b_9 - b_{10}$	$c_{11} = b_{13}/2 - b_{15}$	$c_{11} = b_9 - b_{10}$	*don't care*
	$c_{12} = b_{12} + b_{15}$	$c_{12} = b_{12} + b_{15}$	$c_{12} = b_8 + b_{10}$	$c_{12} = b_{12} + b_{15}$	*don't care*
	$c_{13} = b_{13} + b_{14}$	$c_{13} = b_{13} + b_{14}$	$c_{13} = b_9 + b_{11}/2$	$c_{13} = b_{13} + b_{14}$	*don't care*
	$c_{14} = b_{12} - b_{15}$	$c_{14} = b_{12} - b_{15}$	$c_{14} = b_8 - b_{10}$	$c_{14} = b_{12} - b_{15}$	*don't care*
	$c_{15} = b_{13} - b_{14}$	$c_{15} = b_{13} - b_{14}$	$c_{15} = b_9/2 - b_{11}$	$c_{15} = b_{13} - b_{14}$	*don't care*
ESTÁGIO S	$S_0 = (c_0 + c_1)/2$	$S_0 = c_0 + c_1$	$S_0 = c_0 + c_1$	$S_0 = c_0 + c_1$	$S_0 = c_0 + c_1$
	$S_1 = (c_2 + c_3)/2$	$S_1 = 2*c_2 + c_3$	$S_1 = c_2 + c_3$	$S_1 = c_2 + c_3$	*don't care*
	$S_2 = (c_0 - c_1)/2$	$S_2 = c_0 - c_1$	$S_2 = c_0 - c_1$	$S_2 = c_0 - c_1$	$S_1 = c_0 - c_1$
	$S_3 = (c_2 - c_3)/2$	$S_3 = c_2 - 2*c_3$	$S_3 = c_2 - c_3$	$S_3 = c_2 - c_3$	*don't care*
	$S_4 = (c_4 + c_5)/2$	$S_4 = c_4 + c5$	$S_4 = c_4 + c_5$	$S_4 = c_4 + c_5$	$S_3 = c_4 + c_5$
	$S_5 = (c_6 + c_7)/2$	$S_5 = 2*c_6 + c_7$	$S_5 = c_6 + c_7$	$S_5 = c_6 + c_7$	*don't care*
	$S_6 = (c_4 - c_5)/2$	$S_6 = c_4 - c_5$	$S_7 = c_6 - c_7$	$S_6 = c_4 - c_5$	$S_2 = c_4 - c_5$
	$S_7 = (c_6 - c_7)/2$	$S_7 = c_6 - 2*c_7$	$S_6 = c_4 - c_5$	$S_7 = c_6 - c_7$	*don't care*
	$S_8 = (c_8 + c_9)/2$	$S_8 = c_8 + c_9$	$S_8 = c_8 + c_9$	$S_8 = c_8 + c_9$	*don't care*
	$S_9 = (c_{10} + c_{11})/2$	$S_9 = 2*c_{10} + c_{11}$	$S_9 = c_{10} + c_{11}$	$S_9 = c_{10} + c_{11}$	*don't care*
	$S_{10} = (c_8 - c_9)/2$	$S_{10} = c_8 - c_9$	$S_{10} = c_8 - c_9$	$S_{10} = c_8 - c_9$	*don't care*
	$S_{11} = (c_{10} - c_{11})/2$	$S_{11} = c_{10} - 2*c_{11}$	$S_{11} = c_{10} - c_{11}$	$S_{11} = c_{10} - c_{11}$	*don't care*
	$S_{12} = (c_{12} + c_{13})/2$	$S_{12} = c_{12} + c_{13}$	$S_{12} = c_{12} + c_{13}$	$S_{12} = c_{12} + c_{13}$	*don't care*
	$S_{13} = (c_{14} + c_{15})/2$	$S_{13} = 2*c_{14} + c_{15}$	$S_{13} = c_{14} + c_{15}$	$S_{13} = c_{14} + c_{15}$	*don't care*
	$S_{14} = (c_{12} - c_{13})/2$	$S_{14} = c_{12} - c_{13}$	$S_{14} = c_{12} - c_{13}$	$S_{14} = c_{12} - c_{13}$	*don't care*
	$S_{15} = (c_{14} - c_{15})/2$	$S_{15} = c_{14} - 2*c_{15}$	$S_{15} = c_{14} - c_{15}$	$S_{15} = c_{14} - c_{15}$	*don't care*

No algoritmo da Tabela 4.15 foi inserido um atraso artificial para sincronizar as operações das Hadamards 2x2 direta ou inversa com as operações das transformadas 4x4. O maior inconveniente desta solução é que as entradas de alguns operadores passaram a necessitar de um multiplexador, para selecionar as entradas corretamente, de acordo com o tipo de transformada realizada. Deste modo, como estes multiplexadores estão no caminho crítico da arquitetura, o atraso do cálculo será superior ao da arquitetura paralela apresentada na seção 4.1.4.

O desafio de construir uma arquitetura multitransformada está em adaptar as entradas de alguns dos operadores para receberem diferentes valores, de acordo com o tipo de transformada. Para realizar este controle, foi inserido um multiplexador nas entradas destes operadores. No total, são dez tipos diferentes de possíveis entradas para os operadores, desde a entrada sem uso de nenhum multiplexador, até entradas com uso de multiplexadores de três entradas. A Figura 4.14 apresenta as nove possibilidades de entradas que estão conectadas a multiplexadores. Cabe destacar, na Figuras 4.14 (c), (e) e (h), a presença de um deslocamento para a direita em uma das entradas do multiplexador, indicando as divisões por 2 apresentadas na Tabela 4.15. Do mesmo modo, as Figuras 4.14 (d) e (i) apresentam um deslocamento para a esquerda, indicando uma multiplicação por 2. Finalmente, as Figuras 4.14 (b) e (g) apresentam uma entrada do multiplexador fixa com a constante “0”.

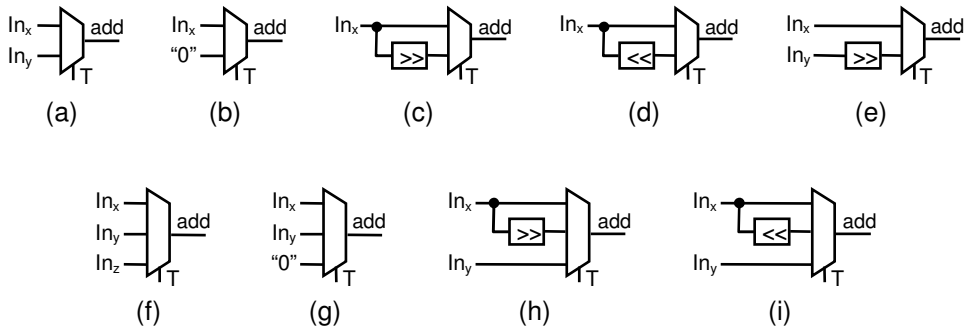

Figura 4.14: Diferentes possibilidades de entradas dos operadores da arquitetura multitransformada com multiparalelismo

A saída da arquitetura multitransformada deve, ainda, ser deslocada para a direita quando a transformada for a Hadamard 4x4 direta. Então, todos os operadores do último nível de pipeline terão um multiplexador adicional, idêntico ao apresentado na Figura 4.14 (c), que entregará para a saída ou o resultado da última operação ou este valor deslocado de uma casa para a direita.

Os multiplexadores e eventuais deslocamentos apresentados na Figura 4.14 são responsáveis pelo acréscimo de lógica no caminho crítico da arquitetura multitransformada em relação à arquitetura de uma única transformada paralela. Mas além dos multiplexadores, os operadores terão uma largura de bits maior do que a arquitetura paralela simples. Isso ocorre porque os operadores de cada estágio da arquitetura devem estar aptos a realizar os cálculos relativos a todas as transformadas sem erros, então, a largura de bits de um determinado estágio da arquitetura multitransformada é definida pela máxima largura de bits dentre todas as transformadas quando consideradas isoladamente.

A soma do acréscimo no caminho crítico causado pelo uso de operadores com maior largura de bits e dos multiplexadores na entrada e saída destes operadores, causa uma

degradação na máxima freqüência de operação atingida, quando se compara a arquitetura multitransformada com as arquiteturas das transformadas simples. Esta questão é mais bem apresentada na próxima seção.

4.5.3 Resultados de Síntese da Arquitetura Multitransformada com Paralelismo Programável

A síntese da arquitetura multitransformada com paralelismo programável foi direcionada para o mesmo FPGA dos experimentos anteriormente apresentados, ou seja, um FPGA Virtex-II Pro VP70 (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada foi a Synplify Pro da Synplicity (SYNPLICITY, 2007). Os resultados de síntese estão apresentados na Tabela 4.16.

Tabela 4.16: Resultados de síntese da arquitetura multitransformada com multiparalelismo

Módulo	Elementos Lógicos	Freqüência (MHz)
Gerenciador de Paralelismo de Entrada	302	201,7
Gerenciador de Paralelismo de Saída	403	201,8
Núcleo da Multitransformada	1815	201,1
Arquitetura Multitransformada	**2594**	**200,5**

Dispositivo 2VP70FF1517-6

Nos resultados de síntese apresentados na Tabela 4.16, o parâmetro que define o número de bits utilizado na entrada da arquitetura foi fixado com o valor oito. Foram utilizados 327 pinos do FPGA e este é um dos maiores problemas desta solução, pois muitos pinos são necessários, mesmo que não sejam utilizados. Por outro lado, a maioria das ferramentas de síntese é capaz de detectar que determinados pinos dos módulos instanciados não estão sendo utilizados e, deste modo, estas ferramentas são capazes de otimizar o processo de síntese, caso o paralelismo seja fixado nas instâncias superiores que utilizam esta arquitetura.

A Tabela 4.17 apresenta os resultados de desempenho da arquitetura proposta, levando em consideração o nível de paralelismo selecionado. Desta tabela é possível perceber que a taxa de processamento da arquitetura pode variar de 200,5 milhões de amostras por segundo até 3,2 bilhões de amostras por segundo, dependendo do nível de paralelismo.

A comparação com as arquiteturas paralelas das transformadas Hadamard 4x4 e 2x2 e DCT diretas, que foram apresentadas na seção 4.3 está apresentada na Tabela 4.18. As arquiteturas das transformadas inversas não foram desenvolvidas isoladamente, por isso não são usadas na comparação.

Tabela 4.17: Resultados de desempenho da arquitetura multitransformada com multiparalelismo

Nível de Paralelismo	Taxa de Processamento (Mamostras/s)
1 amostra por ciclo	200,5
2 amostras por ciclo	401,0
4 amostras por ciclo	801,9
8 amostras por ciclo	1.603, 8
16 amostras por ciclo	3.207,7

Os resultados da Tabela 4.18 indicam que houve um grande aumento no uso dos recursos de hardware na arquitetura multitranformada, em função, principalmente, de quatro fatores principais: (1) inserção dos gerenciadores de entrada e saída para permitir os múltiplos níveis de paralelismo; (2) elevação no número de bits utilizados em cada estágio do pipeline; (3) inserção de multiplexadores e deslocadores nas entradas e saídas dos operadores e (4) aumento na complexidade do controle. A redução na máxima taxa de processamento atingida é outro dado marcante na Tabela 4.18. Esta redução era esperada, em função do aumento da largura de bits dos operadores e da inserção dos multiplexadores no caminho crítico, como discutido na subseção anterior. Ainda assim, a arquitetura consegue atingir uma taxa de processamento superior a 3,2 bilhões de amostras por segundo, mesmo sendo capaz de processar qualquer uma das transformadas diretas ou inversas do padrão H.264/AVC no perfil *main*.

Tabela 4.18: Comparação entre a arquitetura multitransformada com multiparalelismo e as transformadas paralelas

Solução	Elementos Lógicos	Freqüência (MHz)	Taxa de Processamento (Mamostras/s)
Multitransformada com Multiparalelismo	2594	200,5	3.207,7
Hadamard 4x4 Direta	928	303,6	4.857,6
DCT 4x4 Direta	656	319,7	5.115,2
Hadamard 2x2	108	311,4	4.982,4

Para permitir uma melhor comparação com trabalhos relacionados publicados na literatura, foi gerada uma versão *standard-cells* da arquitetura multitransformada com multiparalelismo. A versão *standard-cells* utilizou a tecnologia TSMC 0,35μm e foi gerada a partir da ferramenta Leonardo Spectrum. Os resultados obtidos apontam uma máxima freqüência de operação de 218,7MHz, permitindo uma taxa de processamento superior a 3,4 bilhões de amostras por segundo. A arquitetura em sua versão *standard-cells* utilizou 18.353 *gates*.

A Tabela 4.19 apresenta a comparação entre os resultados obtidos com a arquitetura multitransformada com multiparalelismo e os resultados obtidos a partir de outros trabalhos publicados na literatura. Alguns dos trabalhos considerados nesta comparação utilizam o princípio da multitransformada, como (WANG, 2003; CHENG, 2004; CHEN 2005). Estes trabalhos processam quaisquer das transformadas 4x4 diretas ou inversas, mas a Hadamard 2x2 não é suportada.

Tabela 4.19: Comparação da arquitetura multitransformada com multiparalelismo com trabalhos relacionados

Solução	Tecnologia	Nível de Paralelismo	Número de Portas	Taxa de Processamento (Mamostras/s)
Multitransformada com Multiparalelismo	0.35µ	16	18.353	3.499
Multitransformada com Multiparalelismo	0.35µ	8	18.353	1.749
Cheng (2004)	0.35µ	8	5.745	800
Chen (2005)	0.18µ	8	6.482	800
Multitransformada com Multiparalelismo	0.35µ	4	18.353	874
Wang (2003)	0.35µ	4	6.538	320

É importante destacar que a solução multitransformada com multiparalelismo é a única capaz de processar todas as cinco transformadas previstas pelo perfil *main* do padrão H.264/AVC. Além disso, esta é a única solução que permite uma seleção no nível de paralelismo empregado, variando de 1 até 16 amostras por ciclo.

Os resultados apresentados na Tabela 4.19 indicam que a arquitetura multitransformada com multiparalelismo atingiu, em seu nível máximo de paralelismo, uma taxa de processamento quatro vezes mais elevada do que a mais alta taxa de processamento dentre as soluções investigadas. Considerando o mesmo nível de paralelismo, a arquitetura multitransformada com multiparalelismo atinge mais do que o dobro das taxas de processamento das soluções usadas nas comparações. Por outro lado, a utilização de recursos de hardware foi uma das mais elevadas, em função do maior grau de paralelismo e também da inserção do controle de entrada e saída. Ainda assim, comparando a relação entre o número de *gates* utilizados e a taxa de processamento de todas as soluções é possível perceber que a multitransformada com multiparalelismo utiliza a menor quantidade de hardware para cada milhão de amostras por segundo que é obtida nas taxas de processamento.

Os resultados obtidos para a arquitetura multitransformada com multiparalelismo foram publicados em (AGOSTINI, 2006). Estes resultados foram considerados interessantes, porque esta arquitetura atingiu elevadas taxas de processamento e permite uma enorme flexibilidade, podendo ser utilizada em diferentes projetos de compressores H.264/AVC, com diferentes níveis de paralelismo, além de tornar desnecessário o projeto de arquiteturas distintas e otimizadas para cada uma das cinco transformadas definidas pelo padrão H.264/AVC.

4.6 Arquiteturas Seriais para os Módulos da Quantização Direta e Inversa

Os módulos da quantização direta (módulo Q) e da quantização inversa (módulo Q^{-1}) estão presentes no codificador H.264/AVC, enquanto que apenas o módulo da quantização inversa está presente no decodificador. As próximas seções apresentam os as arquiteturas desenvolvidas para estes módulos, bem como seus resultados de síntese. Em função da maior simplicidade dos módulos da quantização direta e inversa em relação aos demais módulos de um codec H.264/AVC, a solução arquitetural foi desenvolvida com o objetivo de utilizar poucos recursos de hardware e de atingir os requisitos de processamento para vídeos de alto desempenho.

4.6.1 Arquitetura do Módulo Q – Quantização Direta

Na definição da arquitetura do módulo Q, optou-se por eliminar os cálculos de **MF**, **f** e **qbits**, explicados na seção 3.6.5 deste texto, uma vez que estes valores podem ser interpretados como constantes dependentes de **QP**, do fator de escala **PF** e do passo de quantização **Qstep**. Como são 52 valores possíveis para **QP**, são 2 valores possíveis para **PF** e mais 52 valores diferentes para **Qstep**, é possível determinar que são necessárias 156 constantes diferentes para representar todos os valores possíveis para a constante **MF**. Também é possível definir que serão necessários 9 valores diferentes para representar todos os valores possíveis da constante **qbits** e 18 valores para representar todos os valores possíveis da constante **f**. Deste modo, optou-se por realizar antecipadamente os cálculos necessários para a geração de todos os valores de **MF**, **f** e **qbits** e armazenar estes 183 valores como constantes. Assim, além de economizar os recursos de hardware que seriam necessários para realizar todos estes cálculos, a arquitetura ficou mais simples e mais rápida, sendo reduzida a um controle de seleção das constantes, a um multiplicador, a um somador e a um deslocador.

O diagrama em blocos da arquitetura do módulo Q está apresentado na Figura 4.15, onde estão apresentados o multiplicador (X), o somador (+), o deslocador (BS) e o controle de seleção das constantes **MF**, **f** e **qbits**.

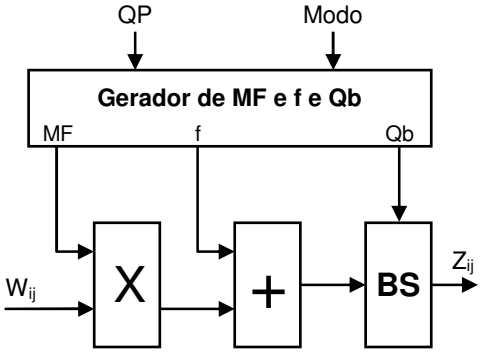

Figura 4.15: Diagrama em blocos da arquitetura do módulo Q serial

O módulo Q foi inicialmente desenvolvido em um pipeline de três estágios. A arquitetura do módulo Q processa um valor de saída do módulo T a cada ciclo de *clock* e, quando o pipeline está cheio, gera um valor quantizado a cada ciclo. A multiplicação utiliza um multiplicador disponível internamente ao FPGA.

A parte de controle do módulo Q é relativamente simples e, a partir do parâmetro de entrada QP, de um sinal externo que indica o início de um novo bloco e de um sinal

externo que indica o modo de predição utilizado, é possível derivar todos os sinais de controle necessários ao correto funcionamento do módulo. O controle foi projetado através de uma máquina de estados finitos, que entrega à parte operativa os sinais de controle necessários aos módulos internos do módulo Q.

4.6.2 Arquitetura do Módulo Q^{-1} – Quantização Inversa

Na quantização inversa, como na quantização direta, as constantes foram previamente calculadas e, neste caso, as constantes foram armazenadas em registradores. Assim, novamente foram economizados recursos de hardware e, em função da simplicidade da arquitetura, o seu desempenho atingiu os requisitos de projeto. Esta arquitetura é formada por um controle de acesso as constantes, um multiplicador, um somador e um deslocador.

O diagrama em blocos da arquitetura do módulo Q^{-1} está apresentado na Figura 4.16 e é similar ao diagrama de blocos da arquitetura do módulo Q, que foi apresentado na Figura 4.15. Na Figura 4.16 estão apresentados o multiplicador (X), o somador (+), o deslocador (BS) e o controle para acesso das constantes.

O módulo Q^{-1}, como o módulo Q, foi desenvolvido em um pipeline de três estágios. A arquitetura do módulo Q^{-1} processa um valor de saída do módulo Q a cada ciclo de *clock* e, quando o pipeline está cheio, gera um valor desquantizado a cada ciclo.

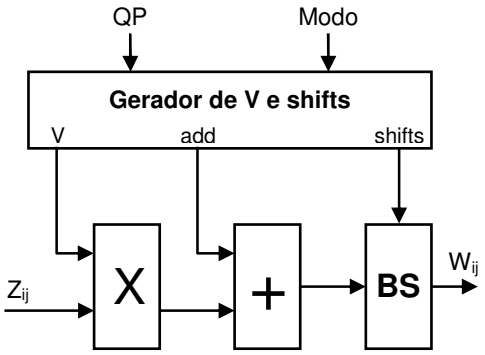

Figura 4.16: Diagrama em blocos da arquitetura do módulo Q^{-1} serial

A parte de controle do módulo Q^{-1} é relativamente simples e, a partir do parâmetro de entrada QP e de um sinal externo que indica o modo de predição utilizado, é possível derivar todos os sinais de controle necessários ao correto funcionamento do módulo. O controle foi projetado através de uma máquina de estados finitos, que entrega à parte operativa os sinais de controle necessários aos módulos internos do módulo Q^{-1}.

4.6.3 Resultados de Síntese dos Módulos Q e Q^{-1}

Os resultados de síntese dos módulos Q e Q^{-1} estão apresentados na Tabela 4.20 e foram direcionados para o FPGA Virtex-II Pro VP70 (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada foi o Synplify Pro da Synplicity (SYNPLICITY, 2007).

Nas descrições VHDL das arquiteturas da quantização direta e inversa, as constantes foram armazenadas em registradores e foram utilizados multiplicadores embarcados do FPGA alvo para realizar a multiplicação presente em cada um destes módulos.

O principal resultado apresentado na Tabela 4.20 diz respeito à máxima taxa de processamento atingida que, para ambos os módulos, ultrapassa a taxa de 93,3 milhões de amostras por segundo, exigido para HDTV, como já foi comentado neste texto. Com

uma folga mínima superior a 32 milhões de amostras por segundo, ambos os módulos estão aptos a integrarem um codec H.264/AVC para resoluções iguais ou inferiores a HDTV. Outro resultado importante diz respeito ao consumo de elementos lógicos, que foi bastante baixo para as duas implementações.

Em função da simplicidade dos módulos da quantização direta e inversa, nenhuma publicação específica sobre estes módulos foi encontrada na literatura. Os resultados da quantização inversa, em conjunto com os resultados do módulo serial das transformadas inversas, foram publicados em (AGOSTINI, 2006c).

Tabela 4.20: Resultados de síntese dos módulos Q e Q^{-1}

Bloco	Elementos Lógicos	Freqüência (MHz)	Taxa de Processamento (Mamostras/s)
Bloco Q - Quantização Direta	324	125,9	125,9
Bloco Q^{-1} - Quantização Inversa	284	226,7	226,7

Dispositivo 2VP70FF1517-6

5 ARQUITETURAS DESENVOLVIDAS PARA A PREDIÇÃO INTER-QUADROS

"Como é linda a liberdade
Sobre o lombo do cavalo
E ouvir o canto do galo
Anunciando a madrugada"

João da Cunha Vargas

Como já foi discutido no capítulo 3 desta tese, o módulo da predição inter-quadros é o mais complexo, tanto no codificador quanto no decodificador. No codificador, a predição inter-quadros é composta por dois módulos: estimação de movimento e compensação de movimento. Já no decodificador, a predição inter-quadros é formada apenas pelo módulo da compensação de movimento.

Este capítulo apresenta as soluções que foram desenvolvidas focando na estimação de movimento, pelo lado do codificador, e na compensação de movimento, por parte do decodificador.

Em relação à estimação de movimento, inicialmente foi desenvolvida uma investigação algorítmica, focando em diversos algoritmos de busca presentes na literatura. Estes algoritmos foram avaliados por diversas métricas, para diferentes condições de execução. Os resultados e uma discussão a respeito desta análise estão apresentados na próxima seção.

Após a investigação algorítmica, três diferentes soluções arquiteturais foram desenvolvidas com base nos algoritmos *Full Search* e sua derivações, com *Pel-Subsampling* 4:1 e com *Pel-Subsampling* 4:1 e *Block-Subsampling* 4:1. Estas soluções focam apenas na busca sobre posições inteiras em apenas um quadro de referência e não consideram múltiplos tamanhos de blocos. Estas soluções foram desenvolvidas com o objetivo de explorar o espaço de projeto e de viabilizar uma solução de estimação de movimento para o H.264/AVC incluindo o maior número possível de ferramentas de codificação previstas neste padrão.

Finalmente, foi desenvolvida uma solução arquitetural para a compensação de movimento do decodificador. Esta solução considerou o perfil *main* do padrão H.264/AVC e é completa, ou seja, implementa todas as ferramentas de codificação previstas pelo padrão.

Todas as arquiteturas desenvolvidas focadas na predição inter-quadros foram descritas em VDHL e sintetizadas para FPGAs Xilinx. O Apêndice A apresenta a metodologia de validação e prototipação das arquiteturas desenvolvidas.

Todas estas investigações sobre o módulo da predição inter-quadros do codificador e decodificador, bem como as soluções desenvolvidas, estão apresentadas nas próximas seções.

5.1 Investigação Algorítmica para a Estimação de Movimento

A investigação algorítmica realizada neste trabalho teve por intenção avaliar alguns dos principais algoritmos de busca para a estimação de movimento, incluindo: *Full Search* (BHASKARAN, 1997; LIN, 2005a), *Three Step Search* (JING, 2004), *One at a Time Search* (RICHARDSON, 2002), *Diamond Search* (KUHN, 1999; YI, 2005), *Hexagon Search* (ZHU, 2002) e *Dual Cross Search* (BANH, 2004). Além disso, o algoritmo *Full Search* foi, também, desenvolvido com subamostragem a nível de pixel (*Pel Subsampling*) (KUHN, 1999) nas relações 2:1 e 4:1 e a nível de bloco (*Block Subsampling*) (KORAH, 2005) junto com a subamostragem a nível de pixel, nas relações 2:1-2:1, 2:1-4:1 e 4:1-4:1. A subamostragem a nível de pixel também foi aplicada aos algoritmos rápidos, com uma relação 2:1. Uma breve descrição destes algoritmos pode ser encontrada no Apêndice B desta tese.

Esta investigação algorítmica foi realizada com o objetivo de dar sustentação ao desenvolvimento arquitetural que se seguiu, como será detalhado no decorrer deste capítulo. Tanto a investigação algorítmica quanto as arquiteturas desenvolvidas consistem de um primeiro esforço para tratar do problema da estimação de movimento. Neste esforço, não foram consideradas diversas ferramentas definidas pelo padrão H.264/AVC, como múltiplos tamanhos de blocos, precisão de ¼ de pixel, vetores apontando para fora do quadro e predição de vetores de movimento. O foco desta investigação reside nas características básicas do processo de estimação de movimento e nos diferentes algoritmos propostos na literatura para realizar esta operação. Com os resultados da investigação algorítmica, espera-se disponibilizar um conjunto de análises que sirvam de apoio à tomada de decisões relativas ao projeto em hardware da ME, mesmo que poucas especificidades do H.264/AVC estejam sendo consideradas neste primeiro momento. Mais algum esforço é necessário para considerar todas as funcionalidades do padrão H.264/AVC e gerar conclusões mais amplas, mas infelizmente este esforço extra não foi concluído a tempo de ser inserido nesta tese.

Por simplicidade de representação, serão usadas abreviaturas para tratar os algoritmos desenvolvidos no decorrer deste texto. Estas abreviaturas estão listadas abaixo, com a descrição do algoritmo que representam.

- FS – *Full Search*;
- DS – *Diamond Search*;
- HS – *Hexagon Search*;
- OT – *One at a Time Search*;
- DCS – *Dual Cross Search*;
- TSS – *Three Step Search*;
- Pel 2:1 – *Pel Subsampling* 2:1;
- Pel 4:1 – *Pel Subsampling* 4:1;
- Bck 2:1 – *Block Subsampling* 2:1;
- Bck 4:1 – *Block Subsampling* 4:1.

As avaliações foram realizadas considerando apenas um quadro de referência e um tamanho fixo de bloco com 16x16 pixels. A resolução dos vídeos de entrada foi definida como SDTV, com 720x480 pixels. Nesta avaliação foram utilizados os 100 primeiros quadros de 10 seqüências de vídeos não comprimidas (VQEG, 2007). As seqüências são bastante variadas, contendo desde vídeos com pouco movimento de objetos e de câmera, até vídeos com muito movimento. A Figura 5.1 ilustra os primeiros quadros de cada um dos 10 vídeos utilizados nas análises.

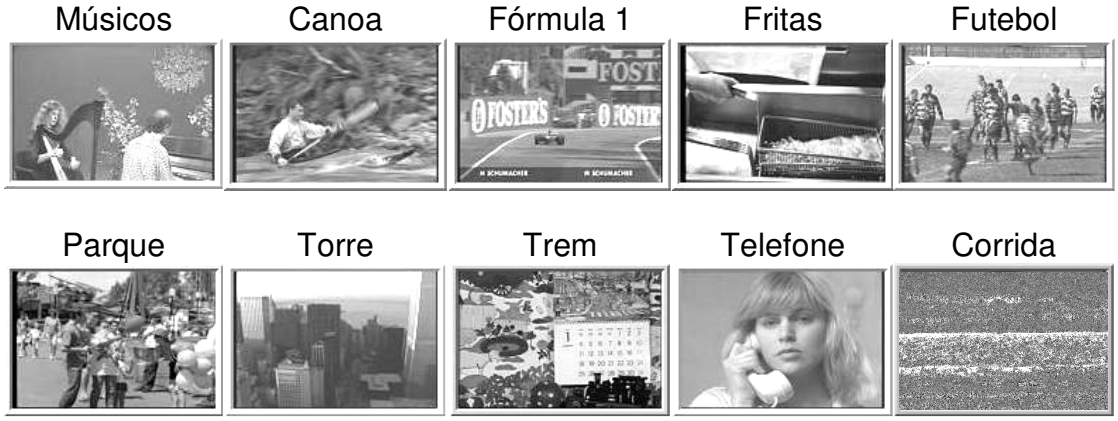

Figura 5.1: Primeiro quadro das dez amostras de vídeo utilizadas nas simulações

As análises consideraram quatro diferentes áreas de pesquisa e o critério de similaridade foi definido como sendo a soma dos erros absolutos (SAD), definido no capítulo 2.

Diversos foram os critérios de comparação considerados nesta análise e foi elevado o número de simulações realizadas. Cada simulação consumiu um tempo entre 4 horas e 60 horas de processamento. No total, foram utilizadas cerca de 2.300 horas de processamento usadas para gerar todos os resultados das simulações. Apenas um pequeno extrato destes resultados está apresentado nesta tese, incluindo o número de cálculos de SAD realizados, a redução do erro e o PSNR gerado por cada algoritmo. Os resultados apresentados foram gerados a partir da média dos resultados obtidos para todas as seqüências de vídeo processadas.

5.1.1 Avaliação do Impacto do Tamanho da Área de Pesquisa

A primeira investigação buscou avaliar a influência do tamanho da área de pesquisa na qualidade dos vetores gerados. Para tanto, alguns experimentos foram realizados utilizando o algoritmo *Full Search*. Nestes experimentos, a área de pesquisa foi definida como o quadro de referência inteiro. Desta forma, todos os vetores encontrados serão os vetores ótimos para esse quadro. Então os vetores gerados foram classificados de acordo com o módulo máximo de suas componentes. Foram definidas 11 diferentes faixas de tamanhos de componentes: 32, 64, 96, 128, 160, 192, 224, 256, 288, 320 e maior que 320. A estimação foi realizada para os 100 primeiros quadros de 5 das 10 seqüências de vídeo apresentadas anteriormente. Este processo é computacionalmente intensivo, sendo que cada seqüência usou mais de 25 horas de processamento em um computador Pentium 4 HT 3GHz, com 1GB de RAM. Dentre as seqüências avaliadas, quatro possuem uma grande quantidade de movimento, o que implica em uma maior probabilidade de vetores com tamanhos grandes de componentes. A última amostra é de

um vídeo com pouco movimento, onde a tendência é de que os vetores possuam componentes de módulo menor.

A Figura 5.2 ilustra a curva percentual do módulo máximo das componentes dos vetores de movimento (MMCVM) para cada uma das amostras de vídeo avaliadas, considerando o tamanho de bloco de 16x16 amostras. Analisando as curvas do gráfico apresentado na Figura 5.2, é possível perceber que a maior parte dos vetores ótimos encontra-se nas primeiras faixas. Nos vídeos de maior movimento ("Canoa", "F1", "Fritas" e "Futebol") existe uma oscilação de 60% a 80% dos vetores ótimos na faixa com tamanho de componente de até 32 pixels. Já para o vídeo "Telefone", que possui menor quantidade de movimento, cerca de 95% dos vetores ótimos estão dentro desta primeira faixa. As curvas ainda apresentam um crescimento considerável, para todas as seqüências, até a faixa com tamanho de componente de até 96 pixels. Após esta faixa, todas as curvas começam a saturar, apresentando um crescimento mínimo no percentual de vetores, sendo que para a amostra "Telefone", o crescimento é de menos de 1% a partir da faixa 96. Para esta investigação, mais de 85% dos vetores ótimos foram encontrados em uma área de pesquisa com vetores com componente de até 96 pixels, para a média obtida com as 5 amostras avaliadas.

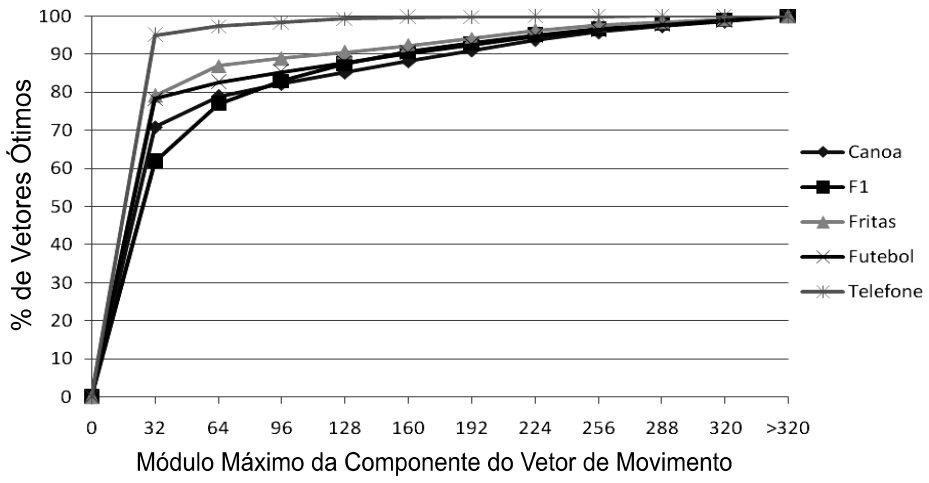

Figura 5.2: Gráfico com percentual de vetores ótimos por faixa de tamanho de vetor

Uma segunda análise foi realizada no intuito de avaliar a qualidade dos resultados quando a área de pesquisa é restrita a cada uma das faixas apresentadas no gráfico da Figura 5.2. Esta análise é importante, pois a escolha de um vetor de movimento sub-ótimo pode ter um impacto mínimo no erro total gerado, após a compensação de movimento. Por exemplo, se um vetor em uma área maior foi escolhido por possuir um SAD de 812, a escolha de um vetor em uma área mais restrita pode conduzir a um SAD de 813, com um impacto mínimo na qualidade final da estimação.

Para este experimento, os tamanhos de componente de vetores foram restritos para as mesmas faixas do primeiro experimento, ou seja: 32, 64, 96, 128, 160, 192, 224, 256, 288, 320 e maior que 320. Para definir qual é a área de pesquisa a partir do tamanho do vetor, basta considerar que, para um vetor de tamanho máximo de componente igual a n, existem + ou – n amostras na vertical e na horizontal em torno do bloco atual. Como o bloco possui 16x16 amostras, então a área de pesquisa será definida como ($2n$+16) x ($2n$+16) amostras. Considerando vetores com componentes de até 32 amostras, isto é, com n=32, a área de pesquisa será de 80x80 amostras.

Para analisar a qualidade do processo de estimação, foi utilizado o PSNR, já definido no capítulo 2. O gráfico apresentado na Figura 5.3 ilustra as curvas de PSNR para cada um dos vídeos avaliados.

Para todas as seqüências de vídeo utilizadas, as curvas do gráfico da Figura 5.3 mostram que existe uma variação muito pequena no PSNR quando a área de pesquisa é ampliada. A amostra "Telefone", que possui pouca movimentação, apresenta um crescimento imperceptível no PSNR. As demais amostras apresentam um leve crescimento no PSNR entre as faixas de vetores de movimento com tamanhos 32, 64 e 96. A partir da faixa de tamanho 96, o crescimento fica muito pequeno e passa a ser praticamente imperceptível nas curvas do gráfico.

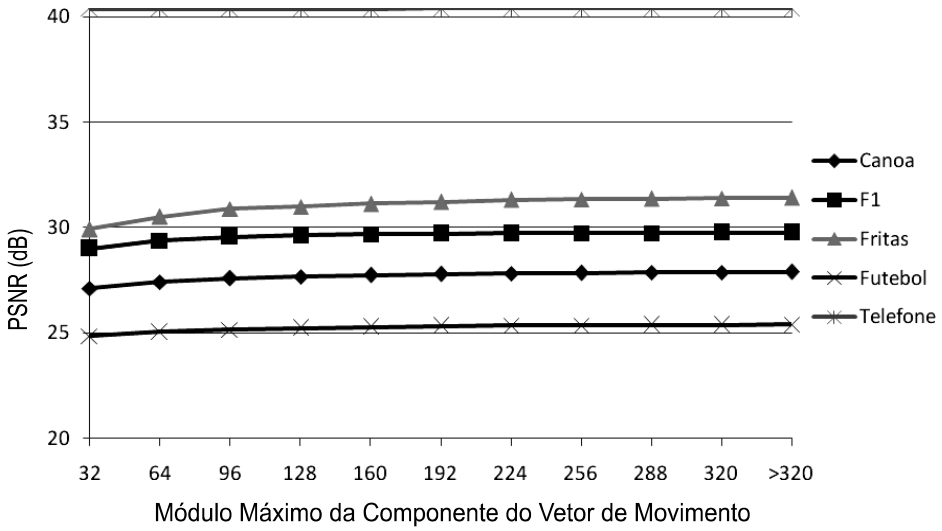

Figura 5.3: Gráfico com curvas de PSNR para todas as amostras avaliadas

A pequena variação no PSNR causada pela ampliação na área de pesquisa indica um pequeno impacto na qualidade do processo de estimação de movimento quando a área de pesquisa é reduzida. O PSNR obtido com vetores com tamanho de até 32 amostras é pouco inferior ao PSNR obtido para áreas mais amplas. Isto ocorre porque a maioria dos vetores encontrados em uma área de pesquisa menor, apesar de não serem os vetores ótimos, geram um resíduo muito pequeno, bem próximo ao gerado pelo vetor ótimo. Esta constatação pode ser verificada no gráfico apresentado na Figura 5.4, que ilustra as curvas de erro para cada uma das amostras de vídeo avaliadas.

O valor de erro é a soma de todos os SADs resultantes do processo de estimação e, quanto menor o valor do erro, melhor será a estimação obtida. Os valores apresentados no gráfico da Figura 5.4 para o erro estão em milhões de unidades. As curvas de erro seguem a mesma tendência das curvas de ganho PSNR, apresentadas no gráfico da Figura 5.3. No entanto, o valor do erro absoluto tende a diminuir mais acentuadamente com o aumento da área de pesquisa. Pode-se perceber que existe uma diminuição significativa no erro entre as faixas de tamanhos de componentes dos vetores de 32, 64 e 96. A partir da faixa de 96, a diminuição do erro passa a ser muito pequena, e sua variação em relação ao aumento da faixa de vetores passa a ser imperceptível.

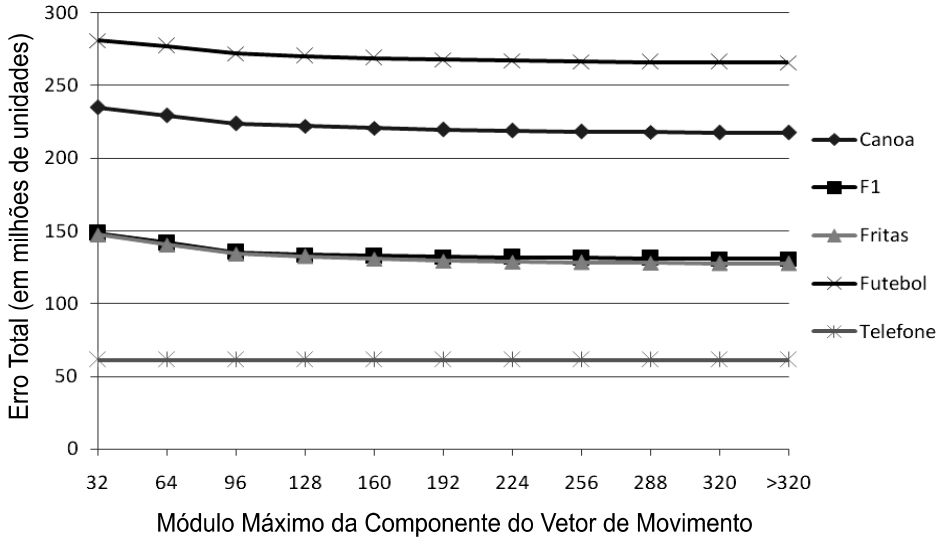

Figura 5.4: Gráfico com curvas de erro para as amostras avaliadas (em milhões de unidades)

Utilizando o algoritmo *Full Search* para realizar a busca é possível determinar quantos cálculos de SAD são necessários para cada área de pesquisa considerada. Conhecendo as dimensões da área de pesquisa, é possível determinar a quantidade de blocos candidatos e, com o número de blocos candidatos, é possível determinar o número de cálculos de SAD que serão realizados. O número de cálculos de SAD é uma boa métrica para avaliar a complexidade computacional de cada solução. Considerando blocos de 16x16 amostras o número de blocos candidatos em uma área de k x k amostras é definido por $(k - 16 + 1)^2$. Tomando como exemplo a área de pesquisa de 80x80, que abrange os vetores com módulo de componentes de até 32 amostras, são 4.225 blocos candidatos por área de pesquisa. Cada bloco candidato possui 16x16 amostras, o que resulta em 256 cálculos de SAD por bloco avaliado. Desta forma, 1.081.600 cálculos de SAD devem ser realizados para avaliar todos os blocos candidatos em uma área de pesquisa de 80x80 amostras. Considerando vídeos SDTV de 740x480 pixels, com blocos de 16x16, são 1.350 blocos que devem ser estimados por quadro. Como uma área de pesquisa deve ser criada para cada bloco, no total, são necessários 1.460.160.000 cálculos de SAD para estimar um quadro com uma área de pesquisa de 80x80 pixels.

O gráfico da Figura 5.5 mostra as curvas de crescimento do número de cálculos de SAD e a diminuição do erro, para cada uma das faixas de vetores. O número de operações está representado em bilhões de operações. Os valores da curva de erro representam a média dos valores de erro obtidos para as cinco amostras avaliadas. Os resultados do erro total foram multiplicados por 100 para que pudessem ser observados mais claramente no gráfico. O crescimento da curva de operações é mais acentuado até a faixa de 224. Isto ocorre, pois para as faixas de vetores maiores que 256, não é possível aumentar as duas dimensões da área de pesquisa na mesma proporção. Para a faixa de vetores de 256, por exemplo, a área de pesquisa resultante é de 528x528, no entanto, este tamanho extrapola o tamanho vertical do quadro, que é de 480. Desta forma, a área de pesquisa efetiva para a faixa de vetores de 256 é de 528x480 pixels. Esta restrição se repete para as faixas de 288, 320 e também para a faixa >320, para a qual a área de pesquisa é o quadro inteiro.

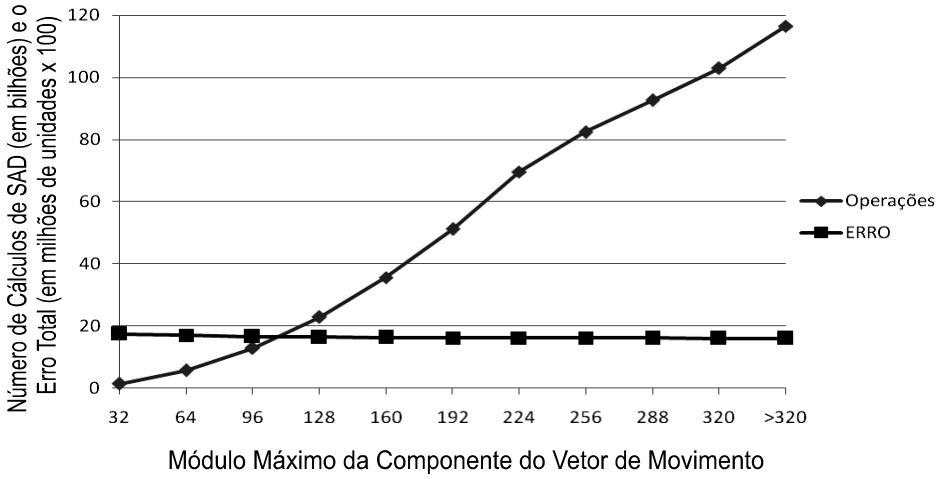

Figura 5.5: Gráfico com número de cálculos de SAD (em bilhões) versus o erro absoluto (em milhões de unidades e multiplicado por 100)

A curva do erro médio mostra a mesma tendência das curvas de erro apresentadas no gráfico da Figura 5.4, sendo que o erro cai entre as faixas de 32, 64 e 96 e, a partir disso, a curva fica praticamente estável. O mais interessante no gráfico da Figura 5.5 é avaliar o número de operações necessárias, que cresce muito rapidamente com o acréscimo das áreas de pesquisa, em contraste a diminuição do erro, que é praticamente imperceptível. O número de operações passa de cerca de 12,8 bilhões de operações, para a faixa de vetores de 96, para cerca de 116,5 bilhões de operações para a faixa >320. Este aumento de aproximadamente 9 vezes no número de operações resulta em um ganho em diminuição de erro de apenas 2,9%. Estes resultados mostram que não é uma estratégia interessante ampliar a área de pesquisa para vetores maiores que 96 quando o tamanho de bloco é 16x16 e a resolução dos vídeos é 740x480 pixels, pois a relação entre diminuição do erro e número de operações necessárias passa a ser muito desfavorável.

É importante ressaltar que os resultados apresentados nesta seção sevem como base para determinar um tamanho de área de pesquisa ótima para o processo de estimação de movimento. O algoritmo utilizado foi o *Full Search*, que sempre encontra o vetor ótimo em uma dada área de pesquisa. Desta forma, o resultado de erro absoluto sempre diminuirá com o aumento da área de pesquisa, mesmo que com ganhos praticamente insignificantes. Estes resultados de erro absoluto gerado e PSNR obtido podem ter comportamentos bem diferentes quando algoritmos rápidos são usados na estimação. Por isso, nas investigações apresentadas nas próximas seções, serão consideradas apenas as áreas de pesquisa 46x46, 80x80, 144x144 e 208x208, com módulos máximos de componentes de 15, 32, 64 e 96, respectivamente. Como a primeira faixa de tamanhos de componentes concentrou um número significativo de vetores ótimos, então, nos próximos experimentos, esta faixa foi dividida em duas e a área de pesquisa de 46x46 amostras (vetores com componentes até 15) foi incluída nas análises. Assim, será possível avaliar com mais detalhes o comportamento dos experimentos para vetores com módulos de componentes menores de 32.

5.1.2 Avaliação da Qualidade dos Resultados

O primeiro critério que será analisado para avaliar a qualidade do processo de estimação será a diminuição do erro em relação ao erro absoluto. O erro absoluto é a soma dos módulos das diferenças, pixel a pixel, entre dois quadros. A Tabela 5.1

apresenta os resultados percentuais de diminuição do erro, para todos os algoritmos investigados, nas quatro áreas de pesquisa avaliadas com seus respectivos módulos máximos das componentes dos vetores de movimento (MMCVM).

Os resultados mostram que o algoritmo *Full Search* (FS) possui uma redução do erro absoluto sempre superior a 51%, para todas as áreas de pesquisa. Para a área de 208x208, a diminuição do erro chega aos 57,4%. Pode-se perceber, também, que a diminuição do erro entre as áreas de 46x46 e 80x80 chega a quase 4%, enquanto que entre as demais áreas, esta diferença é de menos de 1%. Os algoritmos FS com *Pel Subsampling* (2:1 e 4:1) obtiveram a segunda e terceira colocações em termos de redução de erro, sendo que para a versão 2:1, em nenhuma das áreas de pesquisa a diferença na comparação ao algoritmo FS foi maior que 0,4%. Dos algoritmos rápidos, o destaque é para o algoritmo *Diamond Search*, que nas versões sem subamostragem e com subamostragem 2:1, atingiu uma redução média do erro superior a 48%, ficando, no pior caso, apenas 8,3 % abaixo do resultado ótimo gerado pelo *Full Search.* O algoritmo *Full Search* com subamostragem de bloco e de pixel na taxa de 2:1 (FS + Bck 2:1 + Pel 2:1 na Tabela 5.1) atingiu uma redução no erro inferior ao *Diamond Search* para a área de 46x46. Mas a partir da área de 144x144 os seus resultados passam a ser melhores que o do algoritmo DS.

Tabela 5.1: Diminuição percentual do erro absoluto para os algoritmos e áreas de pesquisa investigados

Algoritmos	**Áreas de Pesquisa – MMCVM**			
	46x46 MMCVM=15	**80x80 TMCVM=32**	**144x144 MMCVM=64**	**208x208 MMCVM=96**
FS	51,80	55,70	56,69	57,04
FS + Pel 2:1	51,52	55,40	56,38	56,73
FS + Pel 4:1	47,40	51,17	52,00	52,26
DS	47,15	48,97	49,15	49,17
DS + Pel 2:1	46,74	48,52	48,69	48,71
FS + Bck 2:1 + Pel 2:1	41,91	48,86	52,15	52,51
HS	45,72	47,38	47,55	47,57
HS + Pel 2:1	45,32	46,95	47,10	47,12
FS + Bck 2:1 + Pel 4:1	37,88	47,11	47,94	48,21
FS + Bck 4:1 + Pel 4:1	27,06	45,67	46,51	46,81
OT	42,87	44,50	44,62	44,64
OT + Pel 2:1	41,84	43,33	43,42	43,43
DCS	41,82	43,19	43,31	43,33
TSS + Pel 2:1	40,12	40,54	33,04	40,13
DCS + Pel 2:1	30,19	30,39	30,51	30,52
TSS	37,00	38,88	33,48	31,33

O destaque negativo fica por conta do algoritmo *Three Step Search* (TSS) nas versões com e sem subamostragem, que perderam para todos os demais algoritmos. Além disso, foi o único algoritmo que teve a qualidade do seu resultado reduzida com a ampliação da área de pesquisa. Outra característica curiosa é que, com a aplicação da

subamostragem a nível de pixel, a qualidade dos resultados do algoritmo TSS aumentou, o que é um comportamento atípico e não esperado.

Analisando os dados da Tabela 5.1, é possível concluir que o algoritmo *Full Search* e suas versões com subamostragem a nível de pixel e a nível de bloco, conseguem aproveitar melhor a expansão da área de pesquisa, buscando vetores ótimos em pontos mais distantes, e, conseqüentemente, melhorando seus resultados de erro. Já os algoritmos rápidos possuem uma melhora muito discreta na redução do erro com o aumento da área de pesquisa. Analisando os algoritmos rápidos com os melhores resultados de erro: *Diamond Search* (DS)*, Hexagon Search* (HS)*, One at a Time* (OT) e suas versões com subamostragem, é possível perceber que existe uma redução do erro próxima a 2% entre as áreas de 46x46 e 80x80, e próxima a 0,2% entre as demais áreas.

Os resultados de PSNR para cada um dos algoritmos em todas as áreas de pesquisa avaliadas estão apresentados na Tabela 5.2. Novamente é apresentada a área de pesquisa e o módulo máximo das componentes dos vetores de movimento (MMCVM) para cada caso. A tendência apresentada na Tabela 5.1 se repete na Tabela 5.2, mas há algumas modificações na ordem de classificação dos algoritmos. Esta diferença ocorre, fundamentalmente, porque o critério de similaridade utilizado para gerar os resultados da Tabela 5.1 é o SAD, enquanto que os dados de PSNR apresentados na Tabela 5.2 utilizam o MSE como critério de similaridade. Mais uma vez, o algoritmo *Full Search* e suas versões com *Pel Subsampling* de 2:1 e 4:1 apresentaram os melhores resultados. O algoritmo FS com *Pel Subsampling* 4:1 perde para o algoritmo *Diamond Search* na área de pesquisa de 46x46, mas ganha nas demais áreas apresentadas. Também é interessante notar, na Tabela 5.2, que os resultados de PSNR do algoritmo *Full Search* com subamostragem de bloco e de pixel foram os piores dentre todos os algoritmos.

Tabela 5.2: PSNR dos algoritmos e áreas de pesquisa investigados

Algoritmos	**Áreas de Pesquisa – MMCVM**			
	46x46 MMCVM=15	**80x80 TMCVM=32**	**144x144 MMCVM=64**	**208x208 MMCVM=96**
FS	28,26	28,67	28,90	28,99
FS + Pel 2:1	28,01	28,62	28,86	28,94
FS + Pel 4:1	26,87	27,40	27,55	27,60
DS	27,09	27,26	27,28	27,28
DS + Pel 2:1	27,02	27,19	27,21	27,21
HS	26,81	26,96	26,98	26,98
HS + Pel 2:1	26,75	26,89	26,91	26,91
OT	26,34	26,48	26,50	26,50
OT + Pel 2:1	26,18	26,31	26,32	26,32
DCS	26,19	26,31	26,32	26,32
DCS + Pel 2:1	26,10	26,20	26,22	26,21
TSS + Pel 2:1	25,83	25,64	24,96	25,83
TSS	25,54	25,58	25,01	24,74
FS + Bck 2:1 + Pel 2:1	23,09	24,77	25,00	25,08
FS + Bck 2:1 + Pel 4:1	22,39	23,73	23,88	23,92
FS + Bck 4:1 + Pel 4:1	17,92	20,62	20,76	20,81

Dentre os algoritmos rápidos, deve-se destacar o bom resultado do algoritmo *Diamond Search* (DS) e de sua versão com subamostragem 2:1, que obtiveram os melhores resultados dentre os algoritmos rápidos. Os algoritmos *Hexagon Search* (HS), *One at a Time* (OT) e *Dual Cross Search* (DCS), bem como suas versões com subamostragem, também obtiveram bons resultados. O pior desempenho entre os algoritmos rápidos ficou, novamente, com o algoritmo *Three Step Search* (TSS), que em nenhuma das áreas de pesquisa superou os 25,8dB. Os resultados dos algoritmos rápidos apresentados na Tabela 5.1 se refletem nos seus resultados de PSNR na Tabela 5.2. Houve um crescimento no PSNR entre as áreas de 46x46 e 80x80 amostras, no entanto, entre as demais áreas o crescimento é praticamente nulo, sendo que entre as áreas de 144x144 e 208x208 amostras o crescimento não é percebido com apenas duas casas decimais após a vírgula.

5.1.3 Avaliação do Custo Computacional

O número de cálculos de SAD foi utilizado para medir o custo computacional de cada algoritmo de estimação, para cada uma das áreas de pesquisa. A Tabela 5.3 apresenta os resultados de número de operações de SAD de todos os algoritmos para todas as áreas de pesquisa avaliadas, também considerando os módulos máximos das componentes dos vetores de movimento (MMCVM). Os resultados apresentados na Tabela 5.3 estão em bilhões de operações. Os resultados mostram que o algoritmo *Full Search*, como já era esperado, é o algoritmo com o maior custo computacional, ou seja, ele é o algoritmo que necessita do maior número de cálculos de SAD para realizar a estimação em todas as áreas de pesquisa. O algoritmo *Full Search* necessita de 33,21 bilhões de operações para realizar a estimação de movimento sobre 100 quadros com uma área de pesquisa de 46x46 amostras, e 1287,32 bilhões de operações para os mesmos 100 quadros quando a área de pesquisa cresce para 208x208 amostras. Esta ampliação na área de pesquisa gera um crescimento de, aproximadamente, 39 vezes no número de cálculos de SAD. As versões do FS com *Pel Subsampling* e *Block Subsampling* também possuem um elevado número de cálculos de SAD, que diminuem proporcionalmente ao aumento da subamostragem.

O maior destaque positivo nos dados apresentados na Tabela 5.3 diz respeito aos resultados obtidos com os algoritmos rápidos, que utilizam subamostragem a nível de pixel. Os algoritmos rápidos necessitam de um número muito menor de cálculos de SAD do que o *Full Search*, mesmo quando o FS é usado em conjunto com subamostragem. Quando a subamostragem 2:1 é aplicada aos algoritmos rápidos, estes algoritmos, como esperado, necessitam exatamente a metade dos cálculos de SAD que são necessários para suas versões sem subamostragem.

Outro dado interessante apresentado na Tabela 5.3 é a enorme diferença entre os números de cálculos de SAD necessários para os diversos algoritmos. Comparando os extremos da tabela, o algoritmo *Full Search* utiliza 5.847 vezes mais cálculos de SAD do que o algoritmo OT com subamostragem 2:1, quando a área de pesquisa de 208x208 é considerada.

É interessante a comparação entre os resultados obtidos no número de cálculos de SAD com os resultados de qualidade, apresentados na seção anterior. Desta comparação percebe-se que nem sempre os algoritmos que utilizam uma maior quantidade de cálculos de SAD apresentam os melhores resultados de qualidade da estimação gerada. Os algoritmos FS com *Pel Subsampling* 4:1 e FS com *Block Subsampling* 2:1 e com *Pel Subsampling* 2:1 possuem o mesmo número de operações, no entanto, os resultados de

qualidade do algoritmo FS com *Pel Subsampling* 4:1 são melhores. O algoritmo TSS utiliza o maior número de operações dentre os algoritmos rápidos e apresenta os piores resultados em termos de qualidade da estimação gerada. Já os algoritmos *Diamond Search* (DS), *Hexagon Search* (HS) e *One at a Time* (OT), bem como suas versões com subamostragem, mantêm a tendência de que o uso de mais cálculos de SAD conduz a uma estimação mais qualificada. O algoritmo DS possui os melhores resultados de qualidade e também o maior número de cálculos de SAD dentre os algoritmos rápidos. É importante ressaltar que os algoritmos rápidos podem diminuir o número de cálculos de SAD em até 5.847 vezes, se comparado ao algoritmo *Full Search*, e, mesmo assim, apresentam bons resultados em termos de erro gerado e PSNR. A maior diferença de resultados de redução de erro entre os resultados ótimos, obtidos com o algoritmo *Full Search* e os algoritmos rápidos, obtido com o algoritmo TSS, foi encontrado na área de pesquisa de 208x208, com a redução do erro caindo quase 26%. A menor diferença foi obtida com o algoritmo DS na área de pesquisa de 46x46, onde a redução do erro caiu menos de 8%.

Tabela 5.3: Número de cálculos de SAD (em bilhões de operações) dos algoritmos e áreas de pesquisa investigados

Algoritmos	**Áreas de Pesquisa – TMCVM**			
	46x46 MMCVM=15	**80x80 TMCVM=32**	**144x144 MMCVM=64**	**208x208 MMCVM=96**
OT + Pel 2:1	0,17	0,18	0,19	0,22
DCS + Pel 2:1	0,25	0,26	0,26	0,26
HS + Pel 2:1	0,30	0,32	0,32	0,32
OT	0,34	0,36	0,37	0,44
DS + Pel 2:1	0,41	0,43	0,44	0,44
TSS + Pel 2:1	0,47	0,47	0,47	0,47
DCS	0,50	0,52	0,52	0,52
HS	0,60	0,63	0,63	0,63
DS	0,82	0,86	0,87	0,87
TSS	0,93	0,93	0,93	0,93
FS + Bck 4:1 + Pel 4:1	2,08	9,13	35,95	80,46
FS + Bck 2:1 + Pel 4:1	4,15	18,25	71,89	160,91
FS + Bck 2:1 + Pel 2:1	8,30	36,50	143,78	321,83
FS + Pel 4:1	8,30	36,50	143,78	321,83
FS + Pel 2:1	16,60	73,01	287,55	643,66
FS	33,21	146,02	575,11	1287,32

5.1.4 Análise Focando Implementação da ME em Hardware

As discussões apresentadas nesta seção permitiram avaliar os algoritmos de estimação de movimento investigados considerando, principalmente, a relação existente entre custo computacional e qualidade da estimação. Estas avaliações consideraram dezesseis diferentes algoritmos de busca, dez seqüências de vídeo com resolução SDTV e quatro áreas de pesquisa. O SAD foi usado como critério de similaridade e os blocos foram definidos com tamanho fixo de 16x16 amostras.

As avaliações em diversas áreas de pesquisa mostraram que não é uma estratégia interessante ampliar o tamanho da área para vetores maiores que 96 quando a resolução é SDTV (740x480 pixels) e o tamanho do bloco é de 16x16 amostras. A partir desta área, o custo computacional passa a ser muito elevado e os impactos em qualidade são praticamente nulos.

Considerando a investigação sobre a qualidade da estimação de movimento, é possível concluir que a subamostragem de blocos acima de 2:1 acarreta em um aumento significativamente grande no erro e uma redução no PSNR.

Os resultados para os algoritmos rápidos com e sem subamostragem foram considerados satisfatórios, pois apresentam uma pequena degradação nos resultados de diminuição do erro e de PSNR em relação aos resultados ótimos, mas proporcionam uma redução drástica no número de operações necessárias. Destes algoritmos, cabem destacar o *Diamond Search* e o *Hexagon Search* , bem como suas versões com subamostragem, pois a diminuição da redução do erro causada pelo uso destes algoritmos é a mais próxima do ótimo que foi atingida pelos algoritmos rápidos.

Se o foco é uma implementação em hardware, como é o caso deste trabalho, algumas outras questões devem ser analisadas. Entre elas, cabe ressaltar o nível de paralelismo permitido pelo algoritmo. Como em hardware o paralelismo pode ser explorado com muitos graus de liberdade, pode ser interessante optar por um algoritmo com maior número de cálculos de SADs, mas que não possua dependências de dados entre estes cálculos. Mas é claro que quanto maior a exploração do paralelismo, maior será o consumo de recursos de hardware. Isso pode ser um problema, primeiro porque os recursos de hardware são sempre limitados e também porque se houver mais hardware operando em paralelo, o consumo de energia será mais elevado. Então, é desejável, em um projeto de hardware, utilizar sempre a menor quantidade possível de recursos.

Outra questão relevante está relacionada com a regularidade da arquitetura, pois quanto mais regular é a arquitetura, mas facilmente ela será projetada, uma vez que diversos módulos de hardware poderão ser reusados. Alguns algoritmos são mais propícios do que outros para o desenvolvimento de arquiteturas regulares.

Finalmente, questões como latência, número de interações e sincronismo também são muito relevantes para um projeto em hardware. Para facilitar o sincronismo com outros módulos de um codificador, é interessante que a arquitetura seja embasada em um algoritmo determinístico, isto é, que se saiba com antecedência quantos ciclos são necessários para realizar a estimação. Também é importante que a arquitetura possua uma taxa constante de consumo e produção de dados.

Considerando estas implicações de um projeto em hardware e tendo em mente os resultados das avaliações apresentadas anteriormente, foram desenvolvidas, neste trabalho, arquiteturas para os algoritmos *Full Search*, *Full Search* com *Pel Subsampling* 4:1 e *Full Search* com *Pel Subsampling* 4:1 e *Block Subsampling* 4:1. A decisão de desenvolver arquiteturas com base no algoritmo *Full Search* foi tomada por alguns motivos principais:

1) O paralelismo poderia ser explorado com maior liberdade para atingir os requisitos de tempo da aplicação alvo;

2) O sincronismo com os demais módulos do compressor seria facilitado, pois o algoritmo é determinístico em termos de número de ciclos necessários para realizar o cálculo;

3) As arquiteturas seriam construídas sem um esforço excessivo de projeto, pois seriam basicamente, uma matriz de cálculos de SAD.

Maiores detalhes sobres estas arquiteturas serão apresentados nas próximas seções.

5.2 Arquiteturas Desenvolvidas para a Estimação de Movimento

Esta seção apresenta as três soluções arquiteturais para a estimação de movimento que foram desenvolvidas no escopo desta tese. A primeira delas utiliza o algoritmo *Full Search* e considera uma área de busca de 32x32 amostras. A segunda solução utiliza o algoritmo *Full Search* com subamostragem de pixel na taxa de 4:1 e considera uma área de busca de 64x64 amostras. A terceira e última solução utiliza o algoritmo *Full Search* com subamostragem de pixel e de bloco na taxa de 4:1, também com uma área de busca com 64x64 amostras. Para todos os casos, o tamanho de bloco foi fixado em 16x16 amostras.

5.2.1 Arquiteturas para Estimação de Movimento com Algoritmo *Full Search*

A primeira implementação do módulo da estimação de movimento foi desenvolvida com o objetivo principal de gerar uma solução inicial em hardware para a estimação de movimento, mas que já possuísse desempenho suficiente para processar vídeos de alta resolução. Esta primeira versão da arquitetura não considerou os múltiplos tamanhos de bloco previstos pelo padrão, restringindo-se ao tamanho de bloco 16x16. Também a possibilidade de fazer a estimação sobre posições fracionárias não foi considerada nesta primeira versão. Em função do nível de paralelismo necessário para que a arquitetura atingisse a taxa de processamento necessária para processar vídeos de alta definição e como os recursos de hardware disponíveis são restritos na plataforma de hardware utilizada, a solução para viabilizar esta arquitetura foi reduzir a área de pesquisa para 32x32 amostras. Esta decisão possui um impacto negativo na qualidade dos vetores gerados. Para avaliar melhor este impacto, uma nova simulação foi realizada, considerando os mesmos critérios apresentados na seção 5.1. Esta simulação indicou um PSNR médio de 27,39 dB e uma redução de 46,81% no erro total para vídeos de 740x480 pixels. O PSNR foi reduzido em apenas 1,6 dB em relação a maior das áreas de pesquisa utilizadas nas simulações da seção 5.1, com 208x208 amostras. Considerando ainda esta máxima área de pesquisa, a redução no erro caiu de 57,04% para 46,81%, uma diferença de pouco mais de 10%. Para vídeos SDTV esta área de pesquisa apresenta resultados satisfatórios, mas se a resolução for ampliada para HDTV, por exemplo, estes resultados serão mais deteriorados. Ainda assim optou-se por avançar neste desenvolvimento, para viabilizar uma primeira solução em hardware.

As entradas da estimação de movimento são o bloco do quadro atual e a área de pesquisa do quadro de referência. As informações destes quadros ficam armazenadas na memória externa ao codificador e, quando necessárias, são enviadas para a memória interna da ME. A saída da ME são os vetores de movimento, que são codificados junto com os resíduos da imagem e também são utilizados pela compensação de movimento.

A produção dos vetores de movimento é realizada através da comparação de regiões do quadro atual com regiões do quadro de referência, buscando uma maior semelhança entre elas. Essas regiões são representadas por blocos 16x16 de luminância.

O diagrama de blocos da arquitetura da ME com base no algoritmo *Full Search* está apresentada na Figura 5.6. Vários sinais foram ocultados para permitir uma melhor compreensão da figura. Os detalhes dos principais subsistemas que formam a arquitetura são detalhados nas próximas seções e incluem a linha de cálculo de SAD e as suas unidades de processamento (UPs), o controle, o comparador e o sistema de memória, incluindo as memórias internas e o gerenciador de memória. Os outros módulos incluem um conjunto de registradores para uma linha da área de pesquisa (RLP), um conjunto de registradores para uma linha do bloco do quadro atual (RLB) e seletores dos dados dos registradores (SP e SB).

A memória interna está organizada em seis memórias distintas como está apresentado na Figura 5.6. Duas memórias são destinadas ao armazenamento do bloco do quadro atual e as outras quatro memórias são usadas para armazenar a área de pesquisa.

Os dados da área de pesquisa e do bloco atual que estão armazenados na memória interna são acessados pelo gerenciador de memória (Figura 5.6). Como o bloco está armazenado em duas memórias e a área de pesquisa em outras quatro, quando a ME inicia seu funcionamento, o gerenciador de memória realiza a leitura de uma palavra de cada uma das memórias. Com isso, são acessadas uma linha da área de pesquisa e uma linha do bloco atual. Essas linhas são armazenadas nos registradores da área de pesquisa (RLP na Figura 5.6) e nos registradores de bloco (RLB na Figura 5.6). Nesse momento, o gerenciador envia um sinal que habilita o início das atividades do controle e dos seletores.

Os registradores RLP armazenam uma linha completa da área de pesquisa (32 amostras), utilizando 32 Bytes e os registradores RLB armazenam uma linha do bloco atual, com 16 Bytes.

A unidade de processamento (UP na Figura 5.6) calcula a similaridade (SAD) entre uma palavra do bloco do quadro atual e uma da área de pesquisa. Quatro UPs são combinadas, formando uma linha de SAD. Um conjunto de 17 linhas forma uma matriz de SAD, como está apresentado na Figura 5.6, realizando a busca completa sobre a área de pesquisa do quadro de referência. O controle gerencia as operações e define em que estágio do pipeline cada uma das linhas da matriz de SADs está operando. Depois de finalizado o processamento de toda a região de busca, o comparador recebe os valores de similaridade de todos os blocos candidatos e escolhe aquele com menor valor, gerando o vetor de movimento para este bloco.

A primeira linha de SADs calcula o SAD para os 17 blocos candidatos que iniciam na primeira linha da área de referência e compara o bloco atual a estes blocos candidatos. A segunda linha de SADs calcula o SAD para os 17 blocos que iniciam na segunda linha da área de referência e assim por diante, até que, na décima sétima linha de SADs os SADs dos últimos 17 blocos candidatos são calculados.

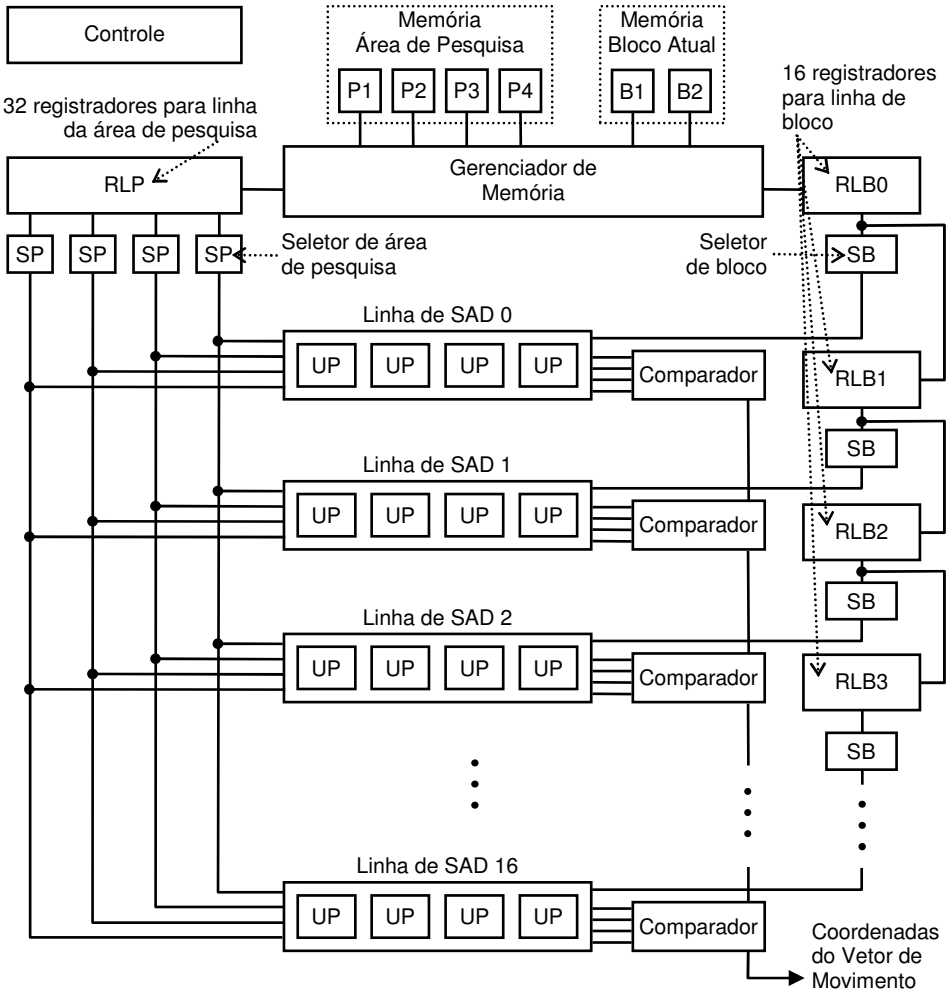

Figura 5.6: Arquitetura da ME com *Full Search* com área de pesquisa de 32x32 amostras

A arquitetura proposta na Figura 5.6, otimiza o compartilhamento das amostras da área de pesquisa e das amostras do bloco do quadro atual sobre o qual se deseja encontrar o melhor casamento e, deste modo, gerar o vetor de movimento. Esta otimização permite diminuir o número de bits de memória necessários e permite reduzir o número de acessos à memória. O cálculo do SAD sempre acontece linha a linha, como será explicado em maiores detalhes na próxima seção. Inicialmente, a primeira linha do bloco atual é usada para calcular o SAD para os blocos candidatos da primeira linha da área de pesquisa. Então, a segunda linha da área de pesquisa é lida da memória da área de pesquisa e a segunda linha do bloco atual é lida da memória do quadro atual. A primeira linha de SAD utiliza estas duas novas informações para calcular a segunda linha de SADs dos blocos que iniciam na primeira linha da área de pesquisa. Mas a segunda linha de SADs irá receber a segunda linha da área de pesquisa e a primeira linha do bloco atual, para iniciar os cálculos dos SADs dos blocos candidatos que começam na segunda linha da área de pesquisa. E assim acontece sucessivamente, até que todos os SADs dos blocos candidatos da área de pesquisa estejam calculados. É por isso que todas as linhas de SAD da Figura 5.6 recebem sempre a mesma linha da área de pesquisa (RLP), mas recebem linhas diferentes do bloco atual (RLB).

A arquitetura apresentada na Figura 5.6 utiliza 160 ciclos de *clock* para preencher as memórias internas e, depois da memória preenchida, um novo vetor é gerado a cada 333

ciclos de *clock*. Isso porque a arquitetura foi construída com um pipeline capaz de paralelizar os cálculos internos relativos a um vetor de movimento, mas não existe um pipeline operacional entre a geração de dois vetores de movimento distintos.

Os módulos principais da arquitetura apresentada na Figura 5.6 serão apresentados em mais detalhes nas próximas seções do texto.

5.2.1.1 Controle da ME com Full Search

O controle da ME foi desenvolvido através de uma máquina de estados finitos que gera, nos instantes de tempo corretos, os sinais de controle necessários aos diversos módulos da arquitetura.

O controle gerencia o pipeline das arquiteturas de cálculo de SAD, habilitando ou desabilitando o funcionamento de cada linha de SAD apresentadas na Figura 5.6. Além disso, o controle é responsável por habilitar os comparadores quando os SADs finais dos blocos candidatos de uma linha de SADs estiverem disponíveis.

O controle também é responsável por gerenciar as escritas nos bancos de registradores (RLP e RLB na Figura 5.6), pois uma linha de bloco utilizada pela linha de SAD em um estágio é útil para as próximas linhas de SAD, como já foi explicado, de forma que tais informações devem ser armazenadas em registradores para serem compartilhadas.

Por fim, é função do controle indicar para as linhas de SAD em qual momento os dados dos registradores RLP e RLB devem ser utilizados.

5.2.1.2 Memórias Internas e Gerenciamento de Memória

As memórias internas utilizadas no estimador de movimento armazenam apenas as informações do componente luminância, pois as informações de crominância não são necessárias para os cálculos do ME, como já foi discutido na seção 3.6.1.

O ME foi projetado utilizando seis memórias internas, como está apresentado na Figura 5.6. Duas memórias armazenam o bloco do quadro atual, e as outras quatro memórias armazenam, cada uma, um quarto da área de pesquisa do quadro de referência. As duas memórias que armazenam o bloco do quadro atual possuem 16 posições, enquanto que as quatro memórias que armazenam a área de pesquisa possuem 32 posições. O ME considera que a transferência entre as memórias internas e os demais módulos internos do ME é realizada através de palavras que possuem oito amostras de luminância da imagem, com 8 bits utilizados para representar cada amostra. Então, cada palavra de memória, para qualquer uma das seis memórias, possui 64 bits. A decisão de utilizar 6 memórias distintas foi, basicamente, função do FPGA escolhido para mapear a arquitetura, que permite memórias com palavras de 64 bits.

O gerenciador de memória na Figura 5.6 é responsável por controlar as leituras de memória de acordo com as necessidades da arquitetura. As amostras lidas das memórias de área de pesquisa são copiadas para os registradores RLP, enquanto que as amostras de bloco atual são armazenadas nos registradores RLB.

O ME acessa seis palavras das memórias (uma de cada memória) a cada 10 ciclos, em um total de 384 bits sendo acessados em intervalos de 10 ciclos. Cada leitura realizada sobre as seis memórias gera uma linha completa da área de pesquisa (com 32 amostras) e uma linha completa do bloco atual (com 16 amostras).

5.2.1.3 Arquitetura para Cálculo do SAD

Como na arquitetura do ME com *Full Search*, foram considerados blocos de 16x16 amostras e uma área de pesquisa de 32x32 amostras, então existem 289 blocos candidatos dentro de cada área de pesquisa. Por isso, é necessário calcular o SAD do bloco com 16x16 amostras para 289 blocos candidatos, em um total de 73.984 cálculos de SAD para realizar a busca completa e encontrar o melhor casamento de bloco em uma área de pesquisa.

A arquitetura para o cálculo do SAD foi construída hierarquicamente. A instância de mais elevado nível hierárquico é a matriz de SADs, que é formada por 17 linhas de SADs, que, por sua vez, são formadas por quatro unidades de processamento (UPs), como está apresentado na Figura 5.6. Cada UP calcula a medida de similaridade da área de pesquisa do quadro de referência com o bloco do quadro atual.

A Figura 5.7 apresenta a arquitetura simplificada de uma UP, onde os sinais de controle estão omitidos. Cada UP recebe como entrada oito amostras do bloco atual (B0 a B7 na Figura 5.7) e oito amostras do bloco candidato (R0 a R7 na Figura 5.7). Isso significa que cada UP calcula paralelamente a similaridade de meia linha do bloco de candidato em relação ao bloco atual. Este nível de paralelismo foi definido para permitir o elevado desempenho necessário ao processamento de vídeos de alta resolução.

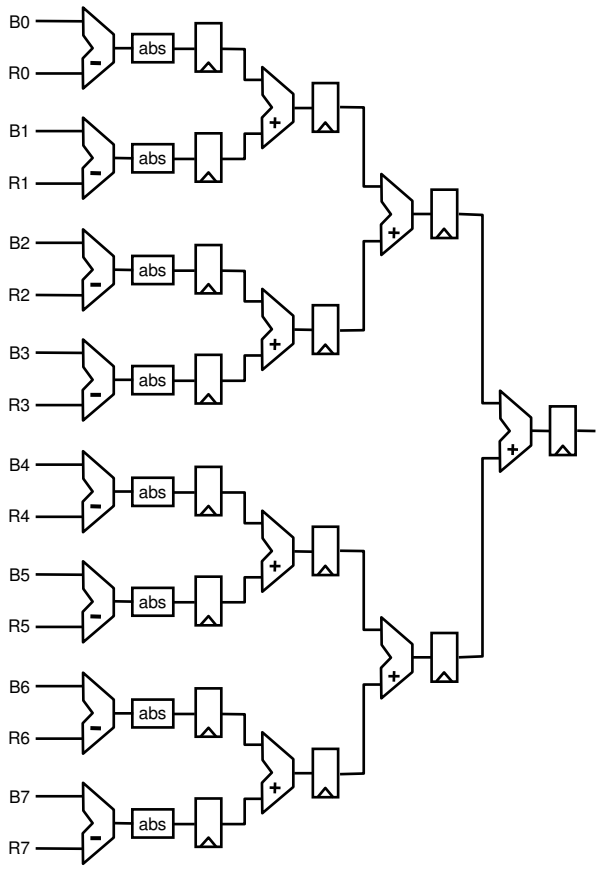

Figura 5.7: Arquitetura de uma UP

As UPs foram projetadas para operar em um pipeline de quatro estágios, como está apresentado na Figura 5.7. Inicialmente, é realizada uma subtração entre as oito amostras do bloco atual (B0 a B7 na Figura 5.7) e as oito amostras do bloco candidato

(R0 a R7 na Figura 5.7). O subtrator gera o resultado em módulo. Então os módulos são somados até gerar um valor único através de uma árvore de somadores. O resultado representa o módulo do erro entre os dois blocos para as oito primeiras amostras, ou seja, para metade da primeira linha dos blocos.

O SAD parcial do bloco candidato (meia linha) gerado pela UP deve ser armazenado e adicionado aos SADs das demais partes do bloco candidato para gerar o SAD total deste um bloco, que é formado por 16 linhas com 16 amostras em cada linha (256 SADs devem ser acumulados).

A linha de SADs, apresentadas na Figura 5.8, agrupa quatro UPs e realiza a acumulação para gerar o valor final do SAD dos blocos candidatos.

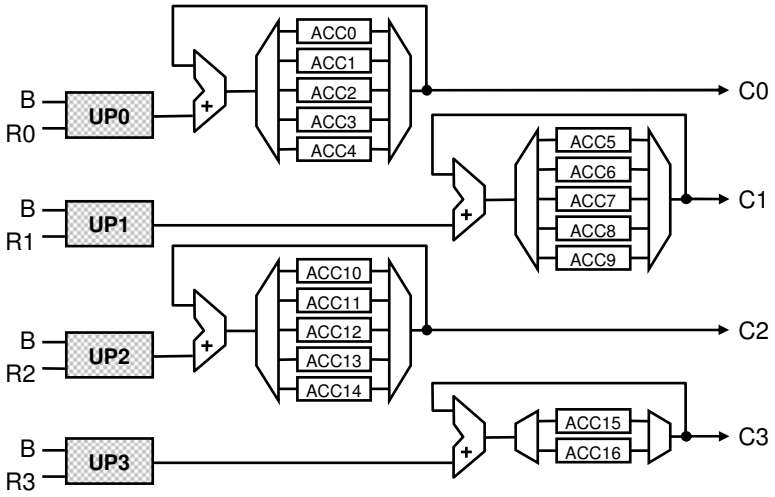

Figura 5.8: Diagrama em blocos de uma linha de SAD

Para aumentar o desempenho e diminuir o consumo de recursos, o paralelismo explorado através do pipeline das UPs não segue a ordem intuitiva. Cada UP é responsável pelo cálculo, em tempos distintos, da similaridade de diferentes blocos candidatos. Com esta solução, as UPs processam os SADs relativos a cinco diferentes blocos candidatos, exceto pela última UP de cada linha de SADs, que processa o SAD de dois blocos candidatos. Então, uma linha de SADs calcula paralelamente (com pipeline) o SAD de 17 diferentes blocos candidatos. Para o cálculo de todos os SADs das linhas dos blocos candidatos, é usada sempre a mesma linha do bloco atual. A Tabela 5.4 apresenta um exemplo das linhas dos blocos candidatos que são processados em cada estágio de pipeline de cada UP de uma linha de SADs.

Como já foi mencionado, cada UP processa metade da linha dos blocos (8 amostras) em cada operação e demora cinco ciclos para concluir este cálculo. A Tabela 5.4 mostra que a arquitetura das UPs calcula os SADs de oito amostras da linha de até cinco blocos (indicados pela letra **a** na Tabela 5.4) para só então retornar ao primeiro bloco e concluir o cálculo sobre a linha, processando as 8 amostras restantes dos blocos (indicados pela letra **b** na Tabela 5.4). Como são necessários mais cinco ciclos para que o cálculo da segunda parte da linha do bloco seja finalizado, então o cálculo completo do SAD de uma linha do bloco é realizado em 10 ciclos.

Tabela 5.4: Exemplo do escalonamento de operações nas UPs de uma linha de SADs

Tempo	Estágio	UP0	UP1	UP2	UP3
0	1	Bloco 0 a	Bloco 5 a	Bloco 10 a	Bloco 15 a
1	2	Bloco 1 a	Bloco 6 a	Bloco 11 a	Bloco 16 a
2	3	Bloco 2 a	Bloco 7 a	Bloco 12 a	-
3	4	Bloco 3 a	Bloco 8 a	Bloco 13 a	-
4	5	Bloco 4 a	Bloco 9 a	Bloco 14 a	-
5	1	Bloco 0 b	Bloco 5 b	Bloco 10 b	Bloco 15 b
6	2	Bloco 1 b	Bloco 6 b	Bloco 11 b	Bloco 16 b
7	3	Bloco 2 b	Bloco 7 b	Bloco 12 b	-
8	4	Bloco 3 b	Bloco 8 b	Bloco 13 b	-
9	5	Bloco 4 b	Bloco 9 b	Bloco 14 b	-

Na Figura 5.8 estão apresentadas as quatro UPs (destacadas em cinza) e estão os acumuladores utilizados para guardar o valor parcial e final dos cálculos do SAD de cada bloco, considerando as suas 17 linhas. O resultado final é gerado depois que as 17 linhas são processadas para cada bloco candidato. Como cada UP processa uma linha do bloco em duas etapas, então cada UP gera 32 resultados parciais de SAD para cada bloco processado. Estes 32 resultados parciais devem ser somados para gerar o SAD final do bloco. Como estes resultados, para um mesmo bloco, não são gerados em paralelo, uma simples estrutura de somador e registrador de acumulação é suficiente para fazer esta totalização. Como as UPs calculam em paralelo o SAD de mais de um bloco, então um registrador é utilizado para armazenar o SAD de cada bloco (ACC0 a ACC16 na Figura 5.8) e um par demultiplexador e multiplexador é necessário para controlar o acesso correto aos registradores, como está apresentado na Figura 5.8. Quando uma linha de SADs concluir seus cálculos, então os registradores ACC0 a ACC16 da Figura 5.8 irão conter os SADs finais dos 17 blocos candidatos.

É importante notar que, para uma UP calcular cinco SADs diferentes (em forma de pipeline), é necessário que, a cada ciclo, oito novas amostras estejam disponíveis para a UP. A seleção dos corretos registradores da linha de bloco e dos registradores da linha da área de pesquisa para cada UP é realizada pelos seletores (SB e SP na Figura 5.6).

5.2.1.4 Arquitetura do Comparador

O comparador foi desenvolvido para realizar quatro comparações de SADs em paralelo, sendo projetado em um pipeline de três estágios. As quatro saídas da linha de SADs (C0 a C3 na Figura 5.8) entregam para o comparador quatro valores de SADs de blocos candidatos em cada ciclo de *clock* e, em cinco ciclos, todos os 17 valores de SAD de uma linha de SAD já foram entregues ao comparador. A arquitetura do comparador deve ser capaz de identificar qual é o menor SAD dentre os 17 SADs da linha de SAD. Quando este menor valor é encontrado, então o comparador compara este valor de SAD com o menor valor dentre os SADs da linha de SAD anterior. A ligação entre os diversos comparadores pode ser observada na Figura 5.6. Então, o menor SAD dentre todos os SADs calculados até então e o vetor de movimento do bloco candidato que gerou este menor SAD, são disponibilizados na saída para o comparador da

próxima linha de SADs ou diretamente para a saída do módulo ME, quando a comparação é realizada na última linha de SAD.

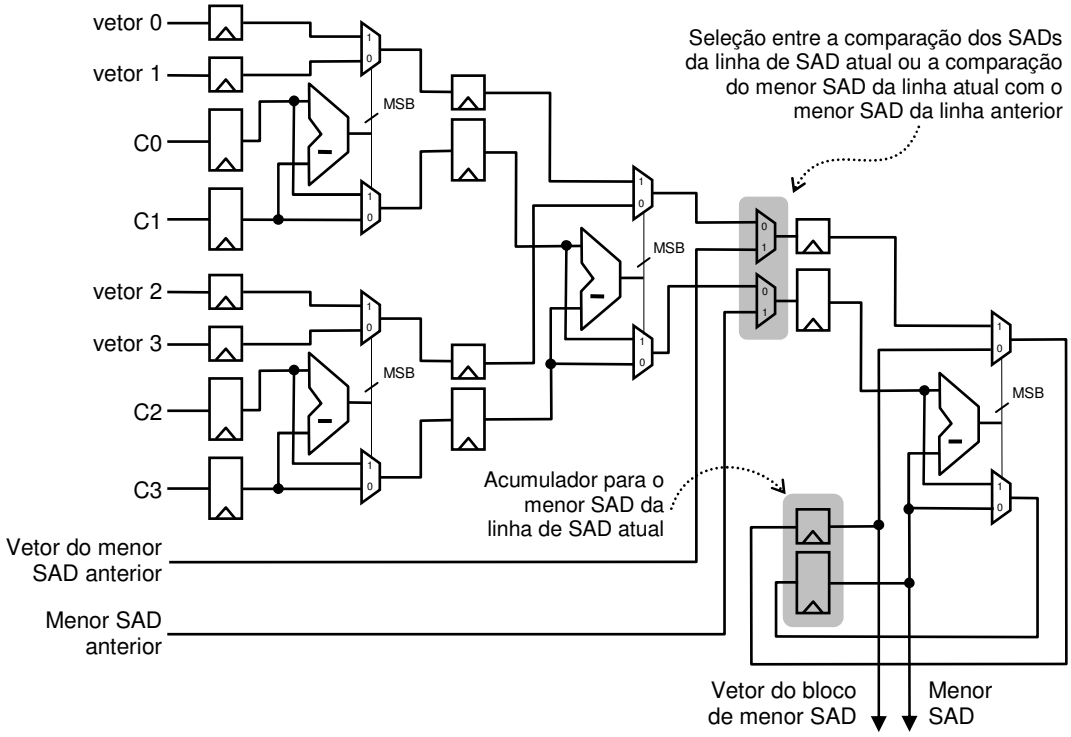

Figura 5.9: Diagrama em blocos do comparador

A Figura 5.9 apresenta a arquitetura proposta para o comparador. Nesta proposta, o comparador é formado, basicamente, por subtratores que subtraem o valor de dois diferentes SADs. Se o resultado é negativo (MSB=1) então o SAD da entrada positiva do subtrator possui uma magnitude menor que o SAD da entrada negativa e, deste modo, o valor da entrada positiva do subtrator deve ser entregue na saída do estágio de comparação. Caso o resultado da subtração seja positivo (MSB=0) então o SAD da entrada negativa do subtrator deve ser entregue na saída. Além dos valores de SAD, o vetor de movimento relativo ao bloco que gerou o menor SAD também deve ser enviado para a saída, para identificar qual bloco gerou o SAD vencedor. Multiplexadores controlados pelo bit de sinal do resultado da subtração selecionam os valores corretos do menor SAD e do respectivo vetor relativo ao bloco de menor SAD. Esta estrutura de comparações com subtratores funciona porque somente valores positivos são permitidos nas entradas. As entradas são SADs de blocos que, no melhor caso, possuem valor igual a zero.

Os multiplexadores destacados em cinza são utilizados quando é realizada a comparação do menor SAD da linha de SAD atual, com o menor SAD da linha de SAD anterior. Esta comparação é sempre a última comparação realizada no comparador e, para ser realizada, o SAD de menor magnitude da linha já deve ter sido encontrado.

Os registradores destacados em cinza na Figura 5.9 armazenam o menor SAD da linha de SAD, armazenando o menor SAD entre o último menor SAD e o SAD atual. Estes registradores funcionam como um acumulador do menor SAD da linha. Após testar os 17 blocos candidatos de uma linha de SAD e encontrar o bloco de menor SAD, os registradores destacados em cinza também armazenam o menor SAD entre o resultado da linha de SADs atual e as linhas de SADs anteriores. Deste modo, o valor

destes registradores é enviado para o comparador da próxima linha de SADs e, se a linha de SADs for a de número 16 (última linha da matriz) então o resultado do vetor de movimento do bloco de menor SAD dentre todos os blocos candidatos é enviado para a saída do estimador de movimento.

5.2.1.5 Resultados de Síntese da ME com Full Search

Os resultados de síntese da ME com *Full Search* estão apresentados na Tabela 5.5 e foram direcionados para o FPGA Virtex-II Pro VP70 (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada neste módulo foi o ISE (XILINX, 2006), também da Xilinx..

Os resultados apresentados na Tabela 5.5 indicam que a estimação de movimento é capaz de operar a uma freqüência máxima de 172 MHz. Considerando que a arquitetura gera um vetor de movimento a cada 333 ciclos de *clock*, a arquitetura mapeada no Virtex-II Pro é capaz de gerar até 516,9 mil vetores de movimento a cada segundo. Como um vetor de movimento, nesta arquitetura, é sempre relativo a um bloco, então é possível afirmar que até 198,5 milhões de amostras são processadas por segundo pela arquitetura da estimação de movimento, pois, no perfil *main* do padrão H.264/AVC um bloco é formado por 384 amostras (256 para Y, 64 para Cb e 64 para Cr). Como a taxa de processamento em amostras por segundo é superior à taxa de processamento das arquiteturas dos módulos T, T^{-1}, Q e Q^{-1} apresentados nas seções anteriores, é possível afirmar que estes módulos poderão ser integrados sem maiores problemas.

Tabela 5.5: Resultados de síntese da ME com *Full Search* e área de pesquisa 32x32

Módulo	Elementos Lógicos	Freqüência (MHz)
Controle	86	253,1
Gerenciamento de Memória	561	305,7
Unidade de Processamento	244	176,1
Comparador	284	276,7
Linha de SAD	1792	172,1
Estimador de Movimento	**37561**	**172,1**

Dispositivo 2VP70FF1517-7

Estes resultados de desempenho apresentados pela arquitetura da estimação de movimento são suficientes para utilizar esta arquitetura em um codificador H.264/AVC direcionado para HDTV. Mesmo com as simplificações na estimação de movimento, como o uso de um único tamanho de bloco, o uso de um único quadro de referência, com a ausência do *quarter* pixel e com uma profundidade de busca de 32 pixels, ainda assim esta arquitetura pode ser utilizada em um codec H.264/AVC. É claro que o uso desta arquitetura em um codec afetaria de forma significativa a eficiência de codificação.

O resultado mais preocupante apresentado na Tabela 5.5 diz respeito ao consumo de elementos lógicos, que foi de 37,5 mil para a arquitetura completa. Este número corresponde a quase 50% dos elementos lógicos disponíveis no FPGA alvo, que é um

dos maiores FPGAs disponíveis na atualidade. Este resultado indica que a estimação de movimento é uma tarefa realmente muito complexa computacionalmente e que o investimento em paralelismo realizado para gerar a taxa de processamento desejada deve ser controlado, senão o consumo de recursos atinge valores impraticáveis. A partir deste resultado e a necessidade de ampliar a área de pesquisa para melhorar o resultado da estimação em vídeos de alta definição, surge a necessidade de buscar soluções alternativas para a estimação de movimento em hardware. Entre estas alternativas pode-se citar o uso de técnicas de *Pel Subsampling* e *Block Subsampling* ou o uso de algum dos algoritmos rápidos investigados na seção 5.1. Nas próximas seções, serão apresentadas duas alternativas arquiteturais, uma com *Pel Subsampling* 4:1 e outra com *Pel Subsampling* 4:1 e *Block Subsampling* 4:1.

5.2.2 Arquitetura para Estimação de Movimento com Algoritmo *Full Search* com *Pel Subsampling* 4:1

Nesta seção, será apresentada a arquitetura proposta para implementar o algoritmo *Full Search* com *Pel Subsampling* 4:1, tomando como referência a arquitetura apresentada na seção 5.2. Utilizando a técnica de *Pel Subsampling,* é possível simplificar a arquitetura, mantendo o desempenho elevado. Além disso, é possível explorar características adicionais, como o aumento na área de pesquisa, que é um ponto fraco da solução apresentada na seção 5.2.

A arquitetura desenvolvida para o algoritmo *Full Search* com *Pel Subsampling* 4:1 utiliza uma área de pesquisa de 64x64 amostras, ao invés da área de 32x32 amostras da arquitetura apresentada na seção 5.2. Esta área foi definida, fundamentalmente, devido às facilidades de adaptação da arquitetura *Full Search* sem subamostragem para a arquitetura desenvolvida com *Pel Subsampling* 4:1. Em uma área de pesquisa de 64x64 amostras existem 2401 blocos candidatos, por isso são necessários 522.496 cálculos de SAD para avaliar todos os blocos candidatos de uma área de pesquisa. Este número de cálculos de SAD é 7,1 vezes maior do que o usado para uma área de pesquisa com 32x32 amostras. Por outro lado, o uso de uma subamostragem a nível de pixel com taxa 4:1 reduz em quatro vezes o número de cálculos de SAD necessários. Então, a arquitetura para o algoritmo *Full Search* com *Pel Subsampling* 4:1 e com área de pesquisa de 64x64 amostras utiliza 130.624 cálculos de SAD para cada área de pesquisa. Assim, esta nova arquitetura usará 1,8 vezes mais cálculos de SAD do que a arquitetura original.

Considerando que se deseja manter o mesmo desempenho da solução anterior, o aumento no número de cálculos de SAD irá ampliar a utilização de recursos de hardware. Como o uso de recursos da arquitetura sem subamostragem já era excessivo, optou-se por realizar modificações visando minimizar este problema. Então, o nível de paralelismo presente nas UPs foi reduzido, como será detalhado nas próximas seções. Esta redução de paralelismo teve impacto no desempenho desta solução, mas ainda assim a arquitetura é capaz de processar vídeos de resolução elevada em tempo real.

Para avaliar melhor os impactos do uso de uma área de pesquisa com 64x64 amostras e do algoritmo *Full Search* com *Pel Subsampling* 4:1, uma nova simulação foi realizada, novamente com os mesmos critérios apresentados na seção 5.1. Esta simulação indicou um PSNR médio de 27,25 dB e uma redução de 50,20% no erro total para vídeos de 740x480 pixels. O PSNR foi reduzido em pouco mais de 0,1 dB em relação à arquitetura sem subamostragem. Por outro lado, a redução no erro cresceu de 46,81% na arquitetura sem subamostragem para 50,20% nesta arquitetura. O resultado

de redução de erro foi melhor do que os obtidos para a arquitetura com *Full Search* e área de 32x32 amostras, mesmo com o uso da subamostragem, porque a ampliação na área de pesquisa teve impacto mais significativo na qualidade do resultado do que o impacto negativo causado pela subamostragem.

A solução arquitetural apresentada nesta seção é embasada na solução proposta na seção 5.2. Assim, apenas as diferenças entre as duas arquiteturas serão discutidas.

O diagrama de blocos da arquitetura da estimação de movimento está apresentada na Figura 5.10. A similaridade da Figura 5.10 com a Figura 5.6, que apresentava a arquitetura *Full Search* com área de pesquisa de 32x32 amostras, demonstra as diversas semelhanças entre as duas soluções. As diferenças estão no número de linhas de SAD, que foi ampliado de 17 para 25, no número de UPs dentro de cada linha de SAD, que foi ampliado de 4 para 5 e no número de memórias usadas para armazenar o bloco atual, que foi reduzida de duas para uma. A Figura 5.10 também mostra o comparador adicional usado na solução *Full Search* com *Pel Subsampling* 4:1. Outras diferenças, como no nível de paralelismo de cada UP, serão detalhados nas próximas seções.

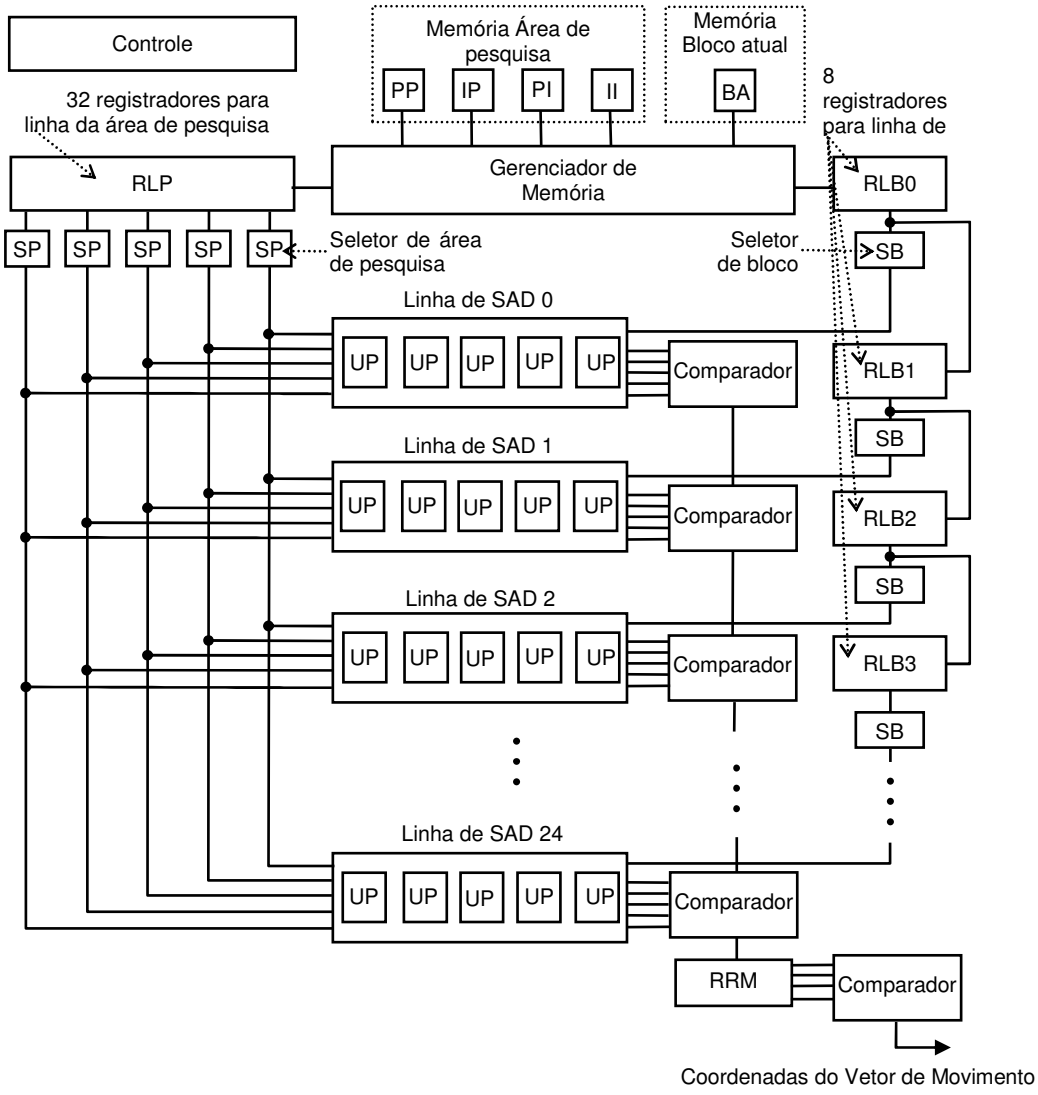

Figura 5.10: Arquitetura da ME com *Full Search* e *Pel Subsampling* 4:1 com área de pesquisa de 64x64 amostras

Em função da redução do nível de paralelismo nas UPs, as linhas de SAD são usadas mais de uma vez para realizar o cálculo do SAD de blocos candidatos distintos. Estas iterações nas linhas de SAD não existiam na arquitetura apresentada na seção 5.2. No total, o processamento foi dividido em quatro etapas e, em cada etapa, uma das quatro memórias da área de pesquisa é utilizada. Os dados de cada memória são completamente processados em 653 ciclos de *clock*. Após encontrar os resultados para as quatro memórias de pesquisa, que foram armazenadas no registrador RRM na Figura 5.10, mais três ciclos são necessários para comparar estes quatro resultados e assim determinar o menor SAD e enviar o vetor correspondente para a saída. Desta forma, 2.615 ciclos de *clock* são necessários para gerar o vetor de movimento para o bloco atual.

Os blocos principais da arquitetura com construções modificadas em relação à arquitetura da seção 5.2 serão apresentados nas próximas seções do texto.

5.2.2.1 Gerenciamento de Memória

A memória interna está organizada em 5 memórias distintas como está apresentado na Figura 5.10, contra as 6 memórias usadas na solução da seção 5.2. Uma memória armazena o bloco do quadro atual e as outras quatro memórias armazenam, cada uma delas, um conjunto distinto de dados da área de pesquisa. Em função da subamostragem de 4:1, a memória que armazena o bloco do quadro atual armazena apenas uma matriz de 8x8 amostras ao invés das 16x16 amostras previstas para o bloco. Esta memória possui oito palavras de 64 bits cada, onde cada posição da memória armazena uma linha do bloco atual, com oito amostras de oito bits cada. Já as quatro memórias da área de pesquisa (PP, PI, IP e II na Figura 5.10) seguem a seguinte divisão:

- PP – Contém os elementos pares das linhas pares;
- IP – Contém os elementos ímpares das linhas pares;
- PI – Contém os elementos pares das linhas ímpares;
- II – Contém os elementos ímpares das linhas ímpares.

Todas as memórias da área de pesquisa possuem 32 palavras de 256 bits cada. Os dados contidos nas quatro memórias de pesquisa representam toda a área de pesquisa de 64x64 amostras. A divisão em quatro memórias foi adotada em função da subamostragem utilizada pelo algoritmo *Pel Subsampling* 4:1, que calcula o SAD para a primeira amostra e desconsidera a amostra seguinte na mesma linha e as duas amostras adjacentes na linha inferior. Desta forma, por exemplo, se o primeiro elemento, posição (0,0) na área de pesquisa, for considerado, o próximo elemento desta linha a ser calculado será o elemento (0,2), e assim sucessivamente, sempre desconsiderando os elementos ímpares desta linha, até que todas as amostras desta linha sejam calculadas. A primeira amostra da próxima linha a ser calculada será a de posição (2,0), ou seja, a linha ímpar deve ser desconsiderada, e assim sucessivamente até que todas as linhas do bloco tenham sido calculadas. Desta forma, com apenas uma leitura da memória podemos calcular os SADs de uma linha para todos os blocos candidatos que comecem em linhas e colunas pares. Assim, ao final dos cálculos para a memória PP, teremos um vetor de movimento resultante para os blocos que começam em linhas e colunas pares.
O mesmo processo é repetido para as outras três memórias, gerando os vetores de movimento resultantes para os blocos candidatos que começam em linhas pares e

colunas ímpares (PI), linhas ímpares e colunas pares (IP) ou linhas ímpares e colunas ímpares (II).

O gerenciador de memória é responsável por controlar as leituras de memória de acordo com as necessidades da arquitetura. As amostras lidas das memórias de área de pesquisa são copiadas para os registradores RLP, enquanto que as amostras de bloco atual são armazenadas nos registradores RLB. Os seletores de linha SP e SB dividem, a palavra lida da memória, de modo que cada UP receba uma informação válida na sua entrada.

A arquitetura acessa duas palavras das memórias, uma da memória do bloco atual e uma da memória da área de pesquisa, a cada 20 ciclos, em um total de 320 bits sendo acessados em intervalos de 20 ciclos.

5.2.2.2 Arquitetura para o Cálculo do SAD

A unidade de processamento (UP na Figura 5.10) calcula a distorção entre ¼ de palavra do bloco do quadro atual e ¼ da palavra do bloco candidato. A decisão de calcular ¼ de palavra na UP, ao invés de ½ da arquitetura apresentada na seção 5.2, foi tomada para reduzir os recursos de hardware utilizados. Calculando ¼ de palavra serão necessárias duas vezes mais interações para o cálculo do SAD do bloco candidato do que na arquitetura da seção 5.2. Isto diminuirá o desempenho da arquitetura, mas, no entanto, proporcionará uma redução significativa nos recursos de hardware utilizados.

Cinco UPs são combinadas para formar uma linha de SAD ao invés das 4 UPs usadas na arquitetura da seção 5.2. O aumento no número de UPs por linha de SAD foi definido em função da elevação da área de pesquisa e da conseqüente elevação no número de blocos candidatos. Cada UP é responsável por processar cinco blocos candidatos, logo, a linha de SAD com cinco UPs é capaz de processar todos os 25 blocos candidatos de uma linha. Um conjunto de 25 linhas forma uma matriz de SAD, como está apresentado na Figura 5.10.

A Figura 5.11 apresenta a arquitetura simplificada de uma UP e é possível perceber que sua estrutura foi simplificada em comparação com a arquitetura da Figura 5.7. Cada UP recebe como entrada duas amostras do bloco atual (B0 e B1 na Figura 5.11) e duas amostras do bloco candidato (R0 e R1 na Figura 5.11). Então, cada UP calcula, em paralelo, a similaridade de ¼ de linha do bloco candidato, em relação ao bloco atual, uma vez que um bloco 16x16, após a subamostragem definida pelo *Pel Subsampling* 4:1, passará a conter apenas 8x8 amostras utilizadas (oito amostras por linha).

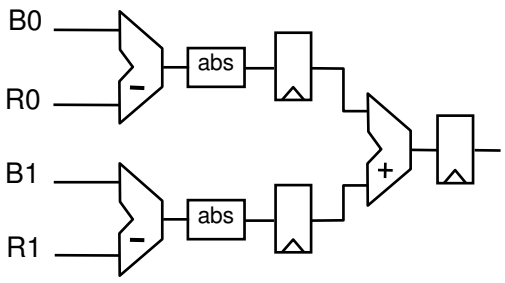

Figura 5.11: Arquitetura simplificada de uma UP

O SAD parcial do bloco candidato gerado pela UP deve ser armazenado e adicionado aos SADs das demais partes do bloco candidato para gerar o SAD total deste bloco, que é formado por oito linhas com oito amostras em cada linha (32 SADs devem ser acumulados). A linha de SADs, apresentada na Figura 5.12, agrupa cinco UPs e realiza a acumulação para gerar o valor final do SAD dos blocos candidatos. Ao todo, são 25 acumuladores, um para cada bloco candidato existente na linha. Cada UP gera valores parciais para cinco acumuladores diferentes. Cada conjunto de cinco acumuladores possui um somador com realimentação, em um total de cinco somadores. A arquitetura apresentada na seção 5.2 possuía apenas 17 registradores e quatro somadores, em função da área de pesquisa menor.

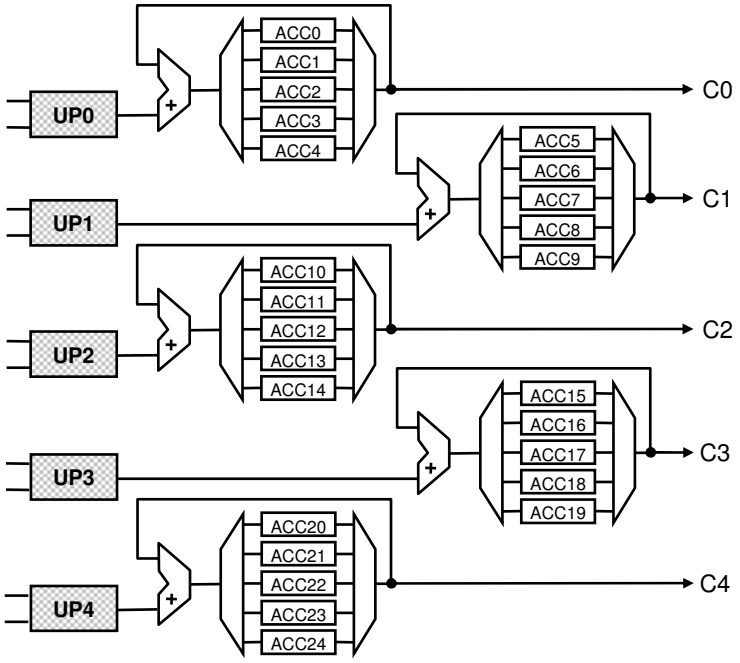

Figura 5.12: Diagrama em blocos de uma linha de SAD modificada

Cada UP é responsável pelo cálculo, em tempos distintos, da similaridade de diferentes blocos candidatos. Com esta solução, as UPs processam os SADs relativos a cinco diferentes blocos candidatos. Então, uma linha de SADs calcula, paralelamente (com pipeline), o SAD de 25 diferentes blocos candidatos. Para o cálculo de todos os SADs das linhas dos blocos candidatos é usada sempre a mesma linha do bloco atual. A Tabela 5.6 apresenta um exemplo das linhas dos blocos candidatos que são processados em cada estágio de pipeline de cada UP de uma linha de SADs. Como já foi mencionado, cada UP processa ¼ da linha dos blocos (2 amostras) em cada operação e usa três ciclos para concluir este cálculo. A Tabela 5.6 mostra que cada UP calcula, em tempos distintos, os SADs de duas amostras da primeira linha de cinco blocos (indicados pela letra **a** na Tabela 5.6) para só então retornar ao primeiro bloco e realizar a segunda etapa do cálculo sobre a linha, processando mais duas amostras destes blocos (indicados pela letra **b** na Tabela 5.6), e assim sucessivamente para a outra metade da linha (indicados pelas letras **c** e **d** na Tabela 5.6).

Tabela 5.6: Exemplo do escalonamento de operações nas UPs de uma linha de SADs.

Tempo	Estágio	UP0	UP1	UP2	UP3	UP4
0	1	Bloco 0 a	Bloco 5 a	Bloco 10 a	Bloco 15 a	Bloco 20 a
1	2	Bloco 1 a	Bloco 6 a	Bloco 11 a	Bloco 16 a	Bloco 21 a
2	3	Bloco 2 a	Bloco 7 a	Bloco 12 a	Bloco 17 a	Bloco 22 a
3	4	Bloco 3 a	Bloco 8 a	Bloco 13 a	Bloco 18 a	Bloco 23 a
4	5	Bloco 4 a	Bloco 9 a	Bloco 14 a	Bloco 19 a	Bloco 24 a
5	1	Bloco 0 b	Bloco 5 b	Bloco 10 b	Bloco 15 b	Bloco 20 b
6	2	Bloco 1 b	Bloco 6 b	Bloco 11 b	Bloco 16 b	Bloco 21 b
7	3	Bloco 2 b	Bloco 7 b	Bloco 12 b	Bloco 17 b	Bloco 22 b
8	4	Bloco 3 b	Bloco 8 b	Bloco 13 b	Bloco 18 b	Bloco 23 b
9	5	Bloco 4 b	Bloco 9 b	Bloco 14 b	Bloco 19 b	Bloco 24 b
10	1	Bloco 0 c	Bloco 5 c	Bloco 10 c	Bloco 15 c	Bloco 20 c
11	2	Bloco 1 c	Bloco 6 c	Bloco 11 c	Bloco 16 c	Bloco 21 c
12	3	Bloco 2 c	Bloco 7 c	Bloco 12 c	Bloco 17 c	Bloco 22 c
13	4	Bloco 3 c	Bloco 8 c	Bloco 13 c	Bloco 18 c	Bloco 23 c
14	5	Bloco 4 c	Bloco 9 c	Bloco 14 c	Bloco 19 c	Bloco 24 c
15	1	Bloco 0 d	Bloco 5 d	Bloco 10 d	Bloco 15 d	Bloco 20 d
16	2	Bloco 1 d	Bloco 6 d	Bloco 11 d	Bloco 16 d	Bloco 21 d
17	3	Bloco 2 d	Bloco 7 d	Bloco 12 d	Bloco 17 d	Bloco 22 d
18	4	Bloco 3 d	Bloco 8 d	Bloco 13 d	Bloco 18 d	Bloco 23 d
19	5	Bloco 4 d	Bloco 9 d	Bloco 14 d	Bloco 19 d	Bloco 24 d

5.2.2.3 Arquitetura do Comparador

O comparador desta arquitetura é muito similar ao comparador utilizado na arquitetura apresentada na seção 5.2. A única diferença está na inserção de uma entrada adicional no comparador, vinda da quinta UP que foi inserida na linha de SAD e que não estava presente na arquitetura da seção 5.2. A Figura 5.13 apresenta a arquitetura modificada do comparador.

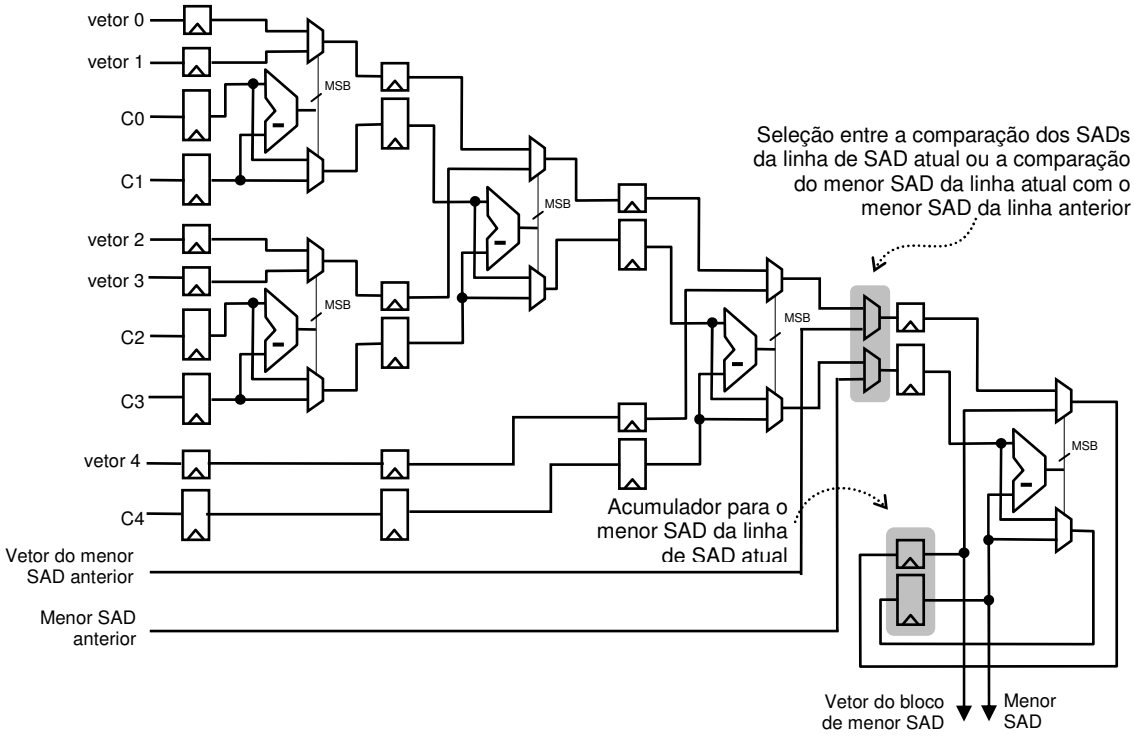

Figura 5.13: Diagrama em blocos do comparador modificado

5.2.2.4 Resultados de Síntese

Os resultados de síntese da ME com *Full Search* e *Pel Subsampling* 4:1 estão apresentados na Tabela 5.7 e foram direcionados para o FPGA Virtex-II Pro VP70 (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada neste módulo foi o ISE (XILINX, 2006), também da Xilinx.

Tabela 5.7: Resultados de síntese da ME com *Full Search* e *Pel Subsampling* 4:1 e área de pesquisa 64x64

Módulo	Elementos Lógicos	Freqüência (MHz)
Controle	157	267,2
Gerenciamento de Memória	605	284,0
Unidade de Processamento	45	381,9
Comparador	235	224,6
Linha de SAD	350	357,0
Matriz de SAD	27.265	173,9
Estimador de Movimento	**31.063**	**123,42**

Dispositivo 2VP70FF1517-7

Um dos resultados mais importantes mostrados na Tabela 5.7 é a freqüência de operação da arquitetura do estimador. Com a freqüência de 123,42MHz, e gerando um novo vetor a cada 2.615 ciclos, a arquitetura do estimador pode gerar mais de 47 mil

vetores de movimento por segundo. Esta taxa implica no processamento de mais de 18 milhões de amostras por segundo.

Os resultados de síntese mostram que arquitetura é capaz de processar até 34,7 quadros SDTV (720 x 480 pixels) por segundo e até 5,8 quadros HDTV (1920x1080 pixels) por segundo.

Esta solução arquitetural foi publicada em (PORTO, 2006), com resultados iniciais do seu desenvolvimento.

5.2.3 Arquitetura para Estimação de Movimento com Algoritmo *Full Search* com *Pel Subsampling* 4:1 e *Block Subsampling* 4:1

Esta seção apresenta a arquitetura desenvolvida para a estimação de movimento utilizando o algoritmo *Full Search* com *Pel Subsampling* 4:1 e também *Block Subsampling* 4:1. Esta arquitetura, apresentada na Figura 5.14, é muito parecida com a solução apresentada na seção 5.3, que não usava *Block Subsampling*

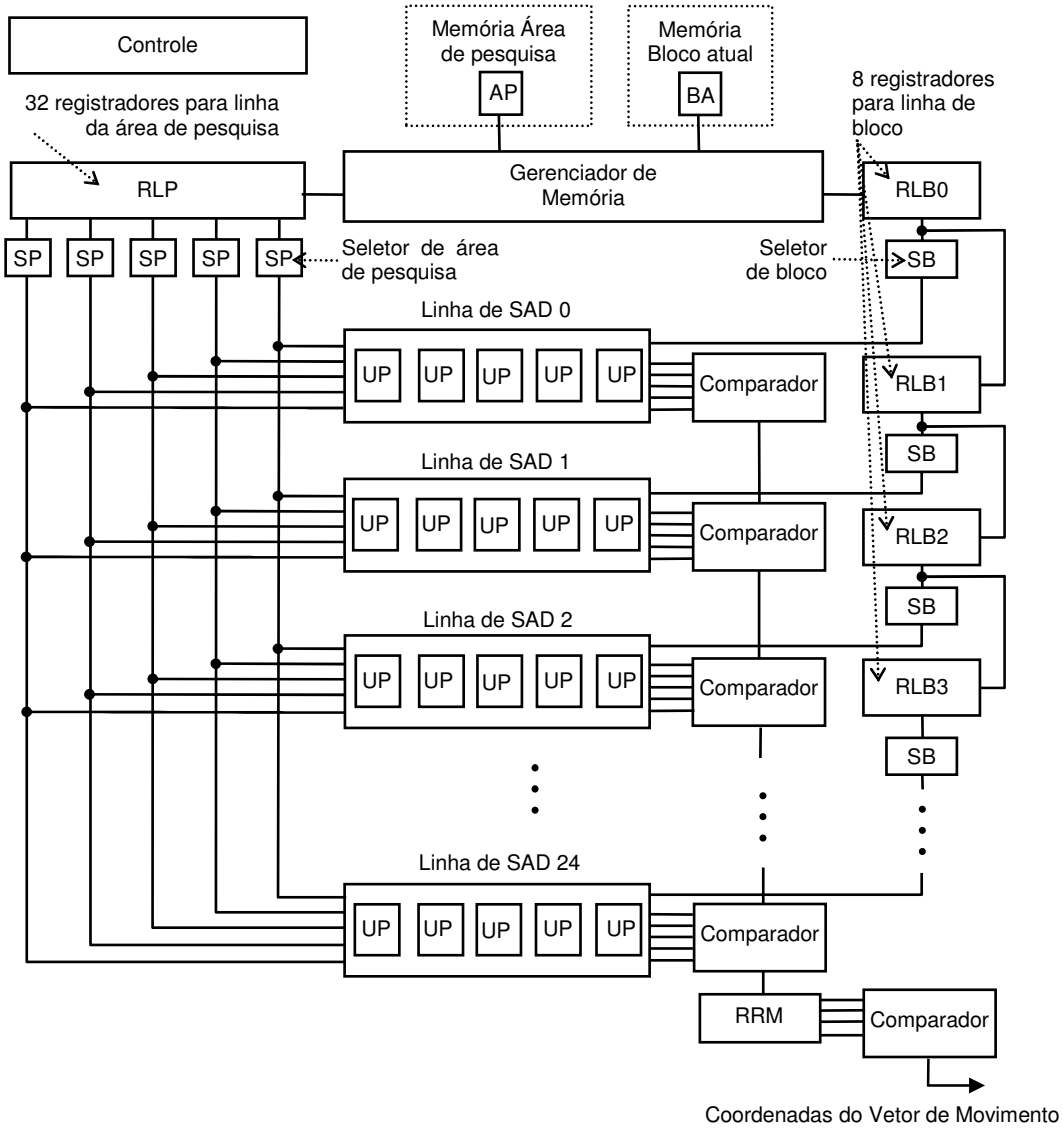

Figura 5.14: Arquitetura da ME com *Full Search*, *Pel Subsampling* 4:1 e *Block Subsampling* 4:1 com área de pesquisa de 64x64 amostras

. Nesta arquitetura, o tamanho de bloco (16x16) e a área de pesquisa (64x64) são as mesmas da arquitetura que usa apenas *Pel Subsampling*. A principal motivação para o desenvolvimento desta solução é a simplicidade de adaptação da arquitetura apresentada na seção 5.3 para utilizar também a técnica de *Block Subsampling*. Estas alterações foram realizadas na eliminação de três das quatro memórias de área de pesquisa presentes na arquitetura anterior, além de simplificações no controle.

Simulações foram realizadas para avaliar o impacto do uso conjunto das técnicas *Pel Subsampling* 4:1 e *Block Subsampling* 4:1. Estas simulações também foram realizadas com os mesmos critérios apresentados na seção 5.1. Os resultados indicaram um PSNR médio de 20,47 dB e uma redução de 44,65% no erro total para vídeos de 740x480 pixels.

Usando as duas técnicas de subamostragem, é possível reduzir sensivelmente o número de cálculos de SAD realizados. O *Block Subsampling* com uma taxa de 4:1 faz com que o número de blocos candidatos em uma área de pesquisa seja dividido por quatro. Então os 2.401 blocos candidatos presentes na arquitetura que usa apenas *Pel Subsampling*, são reduzidos para apenas 601 blocos com o uso do *Block Subsampling*. Como cada bloco é formado por 8x8 amostras ao invés de 16x16 amostras em função do *Pel Subsampling* 4:1, cada bloco candidato usa 64 cálculos de SAD. Assim, são necessários 38.464 cálculos de SAD para avaliar todos os blocos candidatos de uma área de pesquisa. Na arquitetura sem subamostragem e com área de pesquisa 32x32 eram necessários 73.984 cálculos de SAD, enquanto na arquitetura que usa apenas com subamostragem a nível de pixel e com área de pesquisa de 64x64 amostras, o número de cálculo de SADs era de 130.624.

Como pode ser observado na Figura 5.14, a principal diferença desta arquitetura em relação às duas arquiteturas apresentadas anteriormente está no uso de apenas uma memória para a área de pesquisa. Em função do uso da técnica de *Block Subsampling* na taxa de 4:1, três das memórias usadas na arquitetura que usa apenas *Pel Subsampling* puderam ser descartadas, uma vez que os dados lá armazenados não são mais necessários para nenhuma pesquisa. Então, esta memória armazena apenas os elementos pares das linhas pares da área de pesquisa e possui 32 palavras de 256 bits. A eliminação de três memórias tem uma repercussão ainda mais importante, pois as quatro iterações presentes na arquitetura com *Pel Subsampling*, uma para cada memória, também desaparecem e, assim, por eliminar a iteração, a arquitetura deve atingir uma taxa de processamento quatro vezes mais elevada. Considerando as memórias da arquitetura (apresentadas na Figura 5.14) inicialmente preenchidas, um novo vetor é gerado a cada 653 ciclos de *clock*.

A arquitetura para cálculo do SAD desta solução é idêntica à arquitetura apresentada na seção 5.3. A única diferença está na quantidade de iterações necessárias para completar a estimação para uma área de pesquisa completa, que foi reduzida de quatro na arquitetura da seção 5.3 para apenas uma nesta arquitetura. Em função da inexistência das iterações, o projeto da parte de controle foi simplificado em relação ao projeto do controle da arquitetura apresentada na seção 5.3.

5.2.3.1 Resultados de Síntese

Os resultados de síntese da ME *Full Search* com *Pel Subsampling* 4:1 e *Block Subsampling* 4:1 estão apresentados na Tabela 5.8 e foram direcionados para o FPGA

Virtex-II Pro VP70 (XILINX, 2005) da Xilinx e a ferramenta de síntese utilizada neste módulo foi o ISE (XILINX, 2006), também da Xilinx.

Os resultados de síntese mostram que arquitetura é capaz de processar até 72 milhões de amostras por segundo, permitindo o processamento de 139 quadros SDTV (720 x 480 pixels) por segundo e até 23 quadros HDTV (1920x1080 pixels) por segundo.

Tabela 5.8: Resultados de síntese da ME com *Full Search*, *Pel Subsampling* 4:1 e *Block Subsampling* 4:1 e área de pesquisa 64x64

Módulo	Elementos Lógicos	Freqüência (MHz)
Controle	131	278,10
Gerenciador de Memória	56	332,51
Unidade de Processamento	67	341,53
Comparador	235	224,68
Linha de SAD	918	341,53
Matriz de SAD	30.513	143,76
Estimador de Movimento	**31.063**	**123,42**

Dispositivo 2VP70FF1517-7

5.2.4 Comparação entre as Arquiteturas Desenvolvidas e Outros Trabalhos

A Tabela 5.9 mostra os resultados comparativos entre as três arquiteturas de estimação de movimento desenvolvidas neste trabalho. Nos resultados apresentados é interessante perceber que o uso das técnicas de subamostragem, em conjunto com simplificações arquiteturais na UP, conduziu a um consumo menor de elementos lógicos do FPGA alvo, mesmo com a ampliação da área de pesquisa. Em relação ao consumo de memória, o uso da técnica de *Pel Subsampling* reduz apenas o número de amostras do bloco atual que devem ser armazenadas e, então, toda a área de pesquisa deve ser armazenada. Deste modo, como a área de pesquisa foi quadruplicada na solução com *Pel Subsampling*, o número de bits de memória usados cresceu significativamente. Por outro lado, o uso das técnicas de *Pel* e *Block Subsampling* em conjunto reduz a necessidade de armazenamento de dados do bloco atual e da área de pesquisa. Como com *Block Subsampling* 4:1 o número de blocos candidatos é reduzido para a quarta parte dos blocos presentes quando a técnica não é utilizada, então, somente as informações da área de pesquisa referentes aos blocos candidatos válidos devem ser armazenadas em memória. Por isso, o número de bits de memória utilizados é significativamente inferior na arquitetura que usa as duas técnicas de subamostragem.

As freqüências de operação das três arquiteturas não variaram muito, mas o melhor resultado foi obtido com a arquitetura *Full Search* sem uso de subamostragem. Do ponto de vista de taxa de processamento, novamente a solução com *Full Search* apresentou o melhor resultado. Este desempenho superior está embasado em quatro fatos principais: (1) esta arquitetura apresentou a mais elevada freqüência de operação; (2) o paralelismo da UP desta arquitetura é duas vezes maior do que as demais soluções;

(3) esta arquitetura não possui iterações sobre as linhas de SAD, como acontece com a solução com *Pel Subsampling* e (4) esta arquitetura é a que apresenta o menor número de blocos candidatos que precisam ser avaliados.

A questão do número de blocos candidatos é importante, pois mesmo que a subamostragem a nível de blocos reduza o número de blocos a serem avaliados, como a área de pesquisa foi quadruplicada, ainda assim o número de candidatos na solução com *Block Subsampling* 4:1 será maior do que os blocos candidatos presentes na solução com *Full Search* e área de 32x32. Mesmo que cada bloco candidato use quatro vezes menos cálculos de SAD em função do *Pel Subsampling* 4:1, ainda assim todos os blocos devem ser avaliados, passando pelo estágio de comparação e pelas 25 linhas de SAD presentes na arquitetura. Assim, o número de ciclos de *clock* usados até que um vetor seja gerado é maior para as soluções com subamostragem e, deste modo, este aumento no número de ciclos causa uma degradação da taxa de processamento.

Tabela 5.9: Comparação entre as arquiteturas *Full Search*, *Full Search* com *Pel Subsampling* 4:1 e *Full Search* com *Pel Subsampling* 4:1 e *Block Subsampling 4:1*

	Arquitetura		
	Full Search	***Full Search Pel Subsampling* 4:1**	***Full Search Pel Subsampling* 4:1 *Block Subsampling* 4:1**
Área de Pesquisa (amostras)	32 x 32	64 x 64	64 x 64
Blocos Candidatos por Área de Pesquisa	289	2.401	601
Número de Cálculos de SAD por Área de pesquisa	73.984	130.624	38.464
Diminuição no Erro (%)	46,81	50,20	44,65
PSNR (dB)	27,39	27,25	20,47
Elementos Lógicos	37.561	31.063	30.492
Bits de Memória	10.240	33.280	8.704
Freqüência (MHz)	172,10	123,42	123,05
Taxa de Processamento (Mamostras/s)	132	18	72

O PSNR da arquitetura com *Block Subsampling* apresentou uma redução significativa de pouco mais de 7 dB, em relação às soluções sem subamostragem e com *Pel Subsampling*. A redução no erro diminuiu de 50,20% na arquitetura que usa apenas *Pel Subsampling* para 44,65% na solução que usa *Pel* e *Block Subsampling*. Considerando a solução sem subamostragem, a diminuição no erro caiu de 46,81% para 44,65% nesta arquitetura. Em termos de redução de erro, a arquitetura com *Pel* e *Block Subsampling*, ambos em uma taxa de 4:1, apresentou uma piora de 5,5 % em relação à arquitetura que usa apenas *Pel Subsampling*. Como em ambos os casos a área de pesquisa é de 64x64 amostras, então a redução nesta métrica é função exclusiva da taxa de subamostragem de bloco. Comparando esta arquitetura com a arquitetura com *Full*

Search e área de 32x32, foi possível constatar que houve uma queda na redução do erro de pouco mais de 2%. Por outro lado, o PSNR caiu mais 7 dB. É interessante perceber a discrepância nos resultados de redução de erro e de PNSR. Esta discrepância é função, como já explicado nas seções anteriores, da diferença no critério de similaridade usado para o cálculo do erro (SAD) e para cálculo do PSNR (MSE).

Diversos trabalhos relacionados publicados na literatura apresentam arquiteturas que utilizam *Full Search* com ou sem subamostragem. Alguns destes trabalhos foram utilizados nas comparações que serão apresentadas a seguir.

É importante destacar que não foram encontrados trabalhos na literatura com uso de subamostragem e direcionados para FPGAs. Os trabalhos (Huang,2002), (Lee, 2004) e (Chin, 2005) utilizam subamostragem a nível de pixel na taxa de 4:1, mas as arquiteturas foram implementadas em *standard-cells* ao invés de FPGAs. Assim, uma comparação mais efetiva com estes trabalhos é impossível de se realizar

Por outro lado, os trabalhos (Loukil, 2004), (Mohammadzadeh, 2004) e (Roma, 2003) apresentam arquiteturas direcionadas para FPGAs, mas sem o uso de subamostragem.

Estas soluções da literatura consideram uma área de busca de 32x32 amostras, da mesma forma que a arquitetura desenvolvida neste trabalho para o algoritmo *Full Search*. Como as arquiteturas com subamostragem a nível de pixel e de pixel e bloco foram desenvolvidas para uma área de busca de 64x64, foi necessária uma adaptação nos resultados para permitir uma comparação justa. Neste caso, foram calculados os números de ciclos que as arquiteturas com subamostragem de pixel e com subamostragem de pixel com subamostragem de bloco necessitam para gerar um vetor de movimento considerando uma área de busca de 32x32 amostras. Infelizmente, não foi possível encontrar nenhuma arquitetura na literatura com área de pesquisa com 64x64 amostras.

A comparação com estes trabalhos relacionados está apresentada na Tabela 5.10. Esta tabela apresenta a subamostragem realizada em cada uma das arquiteturas, a tecnologia usada, a máxima freqüência de operação atingida e a taxa de processamento. No caso da taxa de processamento, a Tabela 5.10 apresenta o número de quadros HDTV 720p (1.280 x 720 pixels) que podem ser processados por segundo em cada solução.

As arquiteturas desenvolvidas neste trabalho apresentam freqüências de operação muito boas, perdendo apenas para o trabalho de Mohammadzadeh (2004). Ainda assim, as arquiteturas desenvolvidas neste trabalho apresentaram resultados muito bons em termos de taxa de processamento. A arquitetura *Full Search* com *Pel* e *Block Subsampling* atingiu a mais elevada taxa de processamento dentre todas as soluções apresentadas. Esta arquitetura atinge uma taxa de processamento 3,4 vezes maior que a maior taxa de processamento das soluções encontradas na literatura, que é obtida pelo trabalho de Chin (2005). A arquitetura Full Search ficou com a segunda maior taxa de processamento e atinge uma taxa 2,2 vezes maior que o trabalho de Chin (2005). Finalmente, a arquitetura com *Full Search* com *Pel Subsampling* ficou com a quarta maior taxa de processamento e, dentre os trabalhos da literatura, esta solução perde apenas para o trabalho de Chin (2005), atingindo uma taxa de processamento 1,2 vezes menor.

Tabela 5.10. Resultados comparativos para faixa de busca de 32x32 amostras

Solução	Subamostragem	Tecnologia	Freqüência (MHz)	HDTV 720p (quadros/s)
Huang (2002)	Pixel 4:1	0,35 um	50,0	8,75
Lee (2004)	Pixel 4:1	0,35 um	50,0	22,56
Chin (2005)	Pixel 4:1	0,18 um	83,3	63,58
Loukil (2004)	Nenhuma	Altera Stratix	103,8	5,15
Mohammadzadeh (2004)	Nenhuma	Xilinx Virtex-II	191,0	13,75
Roma (2003)	Nenhuma	Xilinx XCV3200e	76,1	20,98
Full Search	Nenhuma	Xilinx Virtex-II Pro	172,1	143,60
Full Search *Pel Subsampling* 4:1	Pixel 4:1	Xilinx Virtex-II Pro	123,4	54,10
Full Search *Pel Subsampling* 4:1 *Block Subsampling* 4:1	Pixel e Bloco 4:1	Xilinx Virtex-II Pro	123,1	216,40

5.3 Arquitetura para Compensação de Movimento do Perfil *Main*

A arquitetura para a compensação de movimento no perfil *main* do padrão H.264/AVC foi desenvolvida visando atingir tempo real quando processando vídeos de alta definição. Esta arquitetura foi desenvolvida na dissertação de mestrado de Arnaldo Azevedo (2006) em conjunto com este trabalho de doutorado. A definição das arquiteturas desenvolvidas naquela dissertação foi realizada de forma conjunta entre o mestrando e o autor desta tese. Em (AZEVEDO, 2006) as arquiteturas propostas estão detalhadas, por isso, nesta tese serão apresentadas apenas as informações mais relevantes sobre este projeto.

A arquitetura da MC foi desenvolvida em um pipeline hierárquico e o mais elevado nível da hierarquia é formado por três módulos principais, como está apresentado na Figura 5.15: preditor de vetores de movimento, acesso à memória e processamento de amostras.

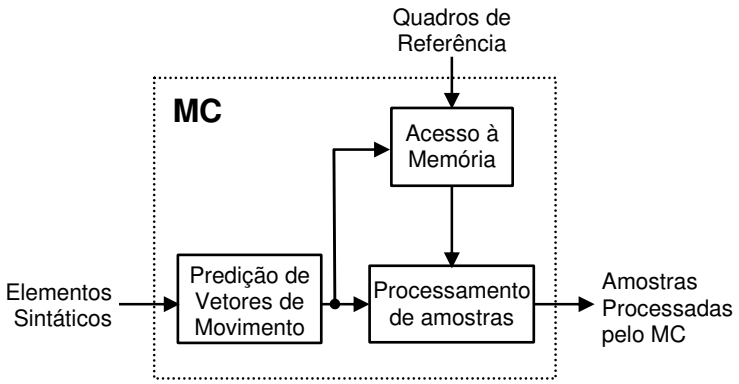

Figura 5.15: Diagrama em blocos da arquitetura do MC

A predição de vetores de movimento lê a informação de macrobloco e seus vetores diferenciais e gera os vetores de movimento para cada macrobloco. Os vetores de

movimento e os índices serão usados para ler da memória a área correta do quadro de referência.

Os quadros previamente decodificados são usados como referências e são armazenados em uma memória externa. Esta memória é organizada em palavras de 64 bits e armazena o quadro de referência em uma ordem *raster*. Cada palavra de 64 bits armazena oito amostras de uma mesma linha do quadro de referência, por isso, a arquitetura processa os dados do quadro de referência linha a linha. A área usada para a compensação de movimento é lida desta memória, mas a posição desta área dentro da memória só será conhecida depois de realizada a predição de vetores de movimento.

O processamento de amostras é responsável por interpolar as amostras e é formado, no perfil *main*, por (ITU-T, 2005): interpolação de pixels para posições fracionárias, predição ponderada, bi-predição e *clipping*.

5.3.1 Arquitetura do Preditor de Vetores de Movimento

A compensação de movimento reconstrói o quadro atual usando, como referências, regiões de quadros previamente decodificados. As regiões de referência são indicadas por vetores de movimento (*Motion Vector* – MV). Estes MVs são calculados através de um processo chamado de predição de vetores de movimento (*Motion Vector Prediction* – MVPr). Existem algumas diferentes opções de predição previstas no MVPr. O modo de predição padrão calcula os MVs usando informações de blocos vizinhos (quando disponíveis) e os vetores de movimento diferenciais lidos do bitstream de entrada. A predição direta usa informações de blocos temporalmente co-localizados de um quadro previamente decodificado. O padrão H.264/AVC prevê dois modos de predição direta para *slices* do tipo B: espacial e temporal. A predição direta espacial examina os macroblocos co-localizados e usa os vetores de movimento dos macroblocos vizinhos. A predição direta temporal aplica um fator de escala para os vetores co-localizados, com base na distância entre o quadro atual e o quadro de referência.

A arquitetura do MVPr foi modelada usando um grupo de registradores e uma máquina de estados finitos, uma vez que o MVPr possui um algoritmo estritamente seqüencial. Dez grupos de registradores foram usados para armazenar as informações dos blocos vizinhos. Vetores de movimento da lista 0 e da lista 1 (ITU-T, 2005) e seus quadros de referência, são calculados em paralelo. A máquina de estados possui 50 estados e suporta predição padrão, predição direta temporal e predição direta espacial.

Simulações mostraram que a arquitetura do preditor de vetores de movimento usa 64 ciclos de *clock* (na média) para processar um macrobloco do tipo P. Para um vídeo codificado com uma seqüência de quadros IPBBP, na média, 127 ciclos de *clock* são usados para processar um macrobloco, quando a predição direta espacial é utilizada, e 88 ciclos de *clock* são usados quando a predição direta temporal é empregada. O MVPr possui uma latência mínima de 17 ciclos de *clock* e uma latência máxima de 226 ciclos de *clock*. Após esta latência, 16 vetores de movimento são sequencialmente enviados para a saída.

5.3.2 Acesso à Memória

No processo de compensação de movimento, algumas áreas dos quadros de referência são usadas muitas vezes para reconstruir os blocos do quadro atual. Os dados relacionados a estas áreas de referência devem ser lidos da memória externa e entregues ao processador de amostras. O reenvio destes dados redundantes requer uma elevada

largura de banda para acesso à memória, por isso, foi desenvolvida uma hierarquia de memória para reduzir os acessos a dados redundantes e, por conseqüência, reduzir a largura de banda da memória.

Para garantir um melhor casamento com este tipo de dados, uma *cache* tridimensional foi desenvolvida. Esta *cache* é indexada pelas posições horizontais e verticais da área a ser acessada no quadro de referência e pelo número do quadro. O número do quadro corresponde à ordem temporal do quadro na seqüência de vídeo e é chamado de *Picture Order Count* (POC).

Os parâmetros de tamanho de *cache* foram determinados através de simulações em software para seqüências de vídeo reais. As simulações foram aplicadas para diferentes conjuntos de dados na *cache* (32, 64 ou 128 conjuntos), com diferentes números de linhas por conjunto (8, 16 ou 32 linhas) e com diferentes números de colunas de por conjunto (24, 40, 72 ou 136 colunas). O tamanho da *cache* foi definido de acordo com a taxa de *misses* obtida.

Considerando os resultados de simulação, foi possível notar que o aumento no número de conjuntos não gera um impacto significante na taxa de *misses*. Então, para reduzir o consumo de hardware, foi escolhida uma *cache* com 32 conjuntos para ser usada junto com a compensação de movimento. O tamanho de cada conjunto foi definido em 40 colunas e 16 linhas.

A saída da *cache* entrega para o processador de amostras, ao mesmo tempo, uma linha de amostras de luminância e duas linhas de amostras de crominância.

Junto com o uso da *cache*, foram utilizadas algumas técnicas para reduzir a largura de banda da memória. Estas técnicas foram adaptadas do que está apresentado em (WANG, 2005a) e são:

1) **Ler somente as amostras necessárias –** Existem casos em que a janela de filtragem não é completamente usada. Se o vetor de movimento possuir um componente vertical igual a zero, então, somente a interpolação horizontal é realizada. Neste caso, somente as amostras na mesma linha são necessárias. O mecanismo de acesso à memória desta arquitetura verifica o vetor de movimento e solicita para a *cache* somente as amostras necessárias.
2) **Intercalar Y, Cb e Cr na memória –** Intercalar as amostras Y, Cb e Cr em uma mesma área de memória reduz a largura de banda. A leitura de memória com o uso da *cache* é determinística, então, é possível intercalar Y, Cb e Cr em uma mesma área de memória. Esta técnica reduz o número de troca de linhas na memória, permitindo leituras em rajadas e evitando consumir os ciclos extra de *clock* que são necessários a cada troca de linha.

O simulador de *cache* foi usado para avaliar a hierarquia de memória desenvolvida. Quatro seqüências de vídeo com resoluções SDTV e HDTV foram testadas. Os vídeos foram codificados com uma seqüência de quadros IPBBPBB e os resultados estão apresentados na Tabela 5.11 e na Tabela 5.12. Os dados apresentados são a média dos resultados obtidos com a utilização destas técnicas para as quatro seqüências de vídeo. Na Tabela 5.11 e na Tabela 5.12, a técnica de ler somente de amostras necessárias está representada como “M1” e a técnica de intercalar Y, Cb e Cr na memória está representada como “M2”.

A Tabela 5.11 apresenta o número de acessos à memória com e sem o uso das otimizações. Estes resultados indicam que as otimizações reduzem em 62% a largura de banda de memória necessária.

Tabela 5.11: Resultados de redução da largura de banda da memória

	Número de Acessos à Memória (x 10^6)	**Redução na Largura de Banda (%)**
Sem Hierarquia	128,63	-
Cache	88,77	30,04%
***Cache* + M1**	47,45	60,49%
***Cache* + M1 + M2**	44,49	62,96%

A Tabela 5.12 apresenta o número de ciclos usados para acessar a memória com e sem as otimizações. Para gerar estes dados, a memória externa foi definida como uma memória SDRAM de 64 bits, com um ciclo de penalidade por troca de linha. A utilização das otimizações causou uma redução em 80% no número de ciclos usados para acessar a memória externa.

Tabela 5.12: Resultados de redução no número de ciclos de acesso à memória

	Número de Ciclos de Acesso à Memória (x 106)	**Redução no número de ciclos (%)**
Sem Hierarquia	273,34	-
Cache	110,96	58,85%
***Cache* + M1**	59,32	76,76%
***Cache* + M1 + M2**	50,42	80,24%

Estes resultados mostram que a hierarquia de memória desenvolvida para o MC causa uma importante redução na largura de banda de memória necessária e no número de ciclos de acesso à memória. Este resultado é importante, pois reduz o gargalo de memória, que é comum neste tipo de aplicação.

5.3.3 Arquitetura do Processador de Amostras

A etapa do processamento de amostras é onde ocorrem transformações nas amostras dentro da compensação de movimento. O processamento das amostras começa depois que o módulo de acesso à memória preenche o *buffer* de entrada com os dados da área do quadro de referência que foram solicitados.

Nesta arquitetura, o processamento de amostras de luminância e crominância ocorre em paralelo. A adoção desta estratégia é necessária para que a arquitetura atinja a taxa de processamento necessária para decodificar vídeos de alta resolução em tempo real. Um *buffer* de 384 posições armazena os resultados de um macrobloco de luminância e

dois macroblocos de crominância e é usado para sincronizar a arquitetura da MC com os demais módulos do decodificador H.264/AVC.

A Figura 5.16 apresenta a arquitetura do módulo de processamento de amostras de luminância. O caminho de dados de luminância é formado por um *buffer* de entrada, um interpolador de quarto de pixel, um preditor ponderado, um *buffer* de 4x4 posições, um cálculo de média para processamento bi-preditivo, um seletor para resultados de predição simples ou de bi-predição e um operador de *clipping*.

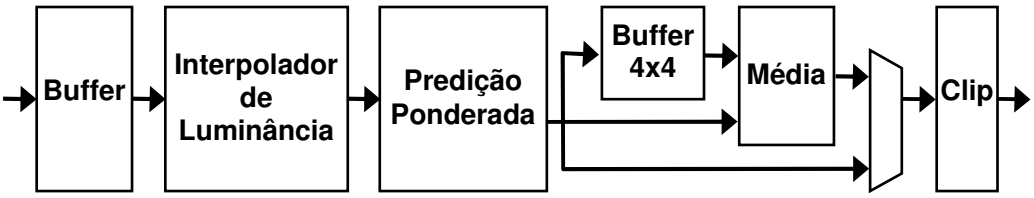

Figura 5.16: Arquitetura do processador de amostras de luminância

O módulo de processamento de amostras de crominância é similar ao processador de amostras de luminância, exceto pelo interpolador e pelo *buffer*. O processamento de amostras de luminância opera sobre blocos de 4x4 amostras, enquanto que o processamento de amostras de crominância opera sobre blocos com 2x2 amostras. O processador de amostras de luminância recebe quatro amostras em paralelo, enquanto que o processador de amostras de crominância recebe duas amostras em paralelo. O processador de amostras completo possui uma latência de 13 ciclos de *clock*. Nas próximas seções serão apresentados detalhes sobre cada um dos módulos principais do processador de amostras.

5.3.3.1 Arquitetura do Interpolador de Luminância

O interpolador de luminância usado nesta arquitetura é inspirado na solução desenvolvida por Wang (2005) e está apresentado na Figura 5.17. Esta solução separa o filtro em duas dimensões em dois filtros em uma dimensão, um aplicado na direção horizontal e outro na direção vertical. A interpolação de ½ pixel é gerada através de um filtro FIR com seis taps (1, -5, 20, 20, -5, 1) e a interpolação em ¼ de pixel é realizada usando um filtro bilinear.

Quatro filtros FIR horizontais e nove verticais foram usados para gerar a interpolação de ½ pixel e quatro filtros bilineares foram usados para gerar a interpolação de ¼ de pixel. Esta arquitetura gera, a cada ciclo de *clock*, quatro amostras com precisão de ¼ de pixel, considerando que os registradores dos filtros horizontais já tenham sido inicializados. A inicialização destes registradores usa cinco ciclos.

Os filtros FIR foram implementados de maneira completamente paralela. Os filtros horizontais foram desenvolvidos com dois estágios de pipeline, enquanto que os filtros verticais não usaram pipeline. A latência do interpolador de luminância é de nove ciclos de *clock*.

Quatro dos nove filtros horizontais processam as amostras com precisão de ½ pixel que foram geradas pelos filtros verticais. Estes quatro filtros horizontais possuem entradas de 14 bits, que podem ser valores negativos. Os quatro filtros horizontais usam multiplicadores presentes no FPGA alvo, enquanto que todos os demais filtros usam somas e deslocamentos ao invés de multiplicadores completos.

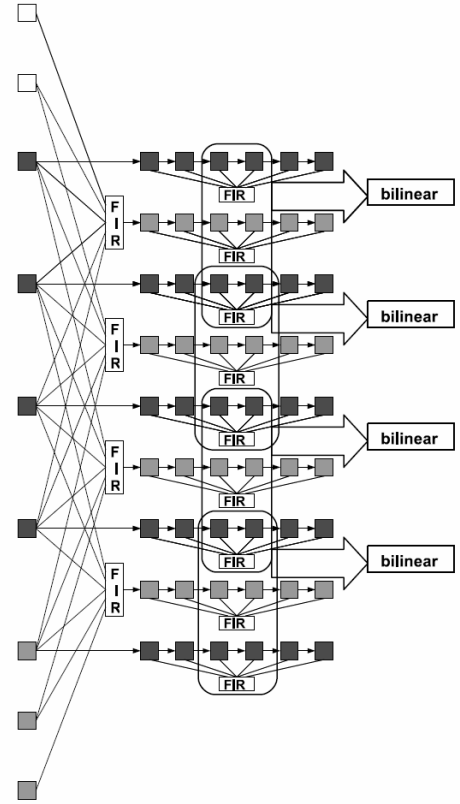

Figura 5.17: Interpolador de luminância

5.3.3.2 Arquitetura do Interpolador de Crominância

O projeto do interpolador de crominância usa um único caminho de dados para processar amostras de crominância Cb e Cr. O processamento de amostras de crominância é composto por um filtro bilinear que executa a interpolação com precisão de 1/8 de pixels. O filtro recebe como entrada uma matriz 3x3 de amostras de crominância (3 amostras por ciclo de *clock*) e gera uma matriz 2x2 de novas amostras interpoladas nas posições fracionárias. A latência para processar um bloco de crominância não bi-preditivo é de quatro ciclos de *clock*.

5.3.3.3 Arquitetura da Predição Ponderada

A função do módulo da predição ponderada é aplicar um fator multiplicativo e um *offset* para cada amostra interpolada. Esta arquitetura usa um módulo para gerar o valor $2^{\log WD-1}$ (necessário ao cálculo da predição ponderada do padrão H.264/AVC (RICHARDSON, 2003)), um multiplicador de 9 bits, um somador e um deslocador. Como a faixa de valores possíveis para $2^{\log WD-1}$ é pequena (de 0 a 7), estes valores foram previamente calculados e armazenados diretamente em registradores.

5.3.3.4 Suporte à Bi-predição no Processamento de Amostras

A arquitetura do MC processa um bloco de cada lista em série para realizar a predição bi-preditiva sem duplicar as arquiteturas de interpolação e de predição ponderada. Ambas as áreas dos quadros de referência são armazenados em um *buffer*. A área de referência da lista 0 é processada e seus resultados são salvos em um *buffer*. Na arquitetura do processador de amostras de luminância, este *buffer* é formado por 4x4 posições, enquanto que na arquitetura do processador de amostras de crominância, este

buffer é formado por 2x2 posições. Quando a predição ponderada gera os resultados interpolados da área de referência da lista 1, estes resultados são enviados para o cálculo de média. Então, os resultados interpolados da área da lista 0, que estão armazenados no *buffer*, são também enviados para o cálculo de média. A média destes dois valores é calculada e este resultado é enviado para a operação de *clipping*.

O interpolador de amostras de luminância usa 19 ciclos de *clock* para processar cada bloco bi-preditivo. Um ciclo é usado para inicializar os cálculos, nove ciclos são usados para os cálculos relativos à lista 0 e outros nove ciclos são usados para os cálculos da lista 1.

5.3.4 Resultados de Síntese

A arquitetura do MC foi sintetizada para o FPGA XC2VP30-7 da família Virtex-II Pro da Xilinx. A síntese foi realizada com a ferramenta Synplicity Synplify 8.1 Pro (SYNPLICITY, 2007). A Tabela 5.13 apresenta os resultados de síntese para os principais módulos da arquitetura: preditor de vetores de movimento, acesso à memória e processamento de amostras.

Os resultados de síntese indicam que a freqüência máxima de operação desta arquitetura é 100 MHz. Com esta freqüência, a arquitetura da MC é capaz de processar quadros HDTV em tempo real. A 100MHz, a arquitetura da MC decodifica até 36,4 quadros HDTV totalmente bi-preditivos (B) por segundo ou até 63,8 quadros preditivos (P) por segundo.

Tabela 5.13: Resultados de síntese da arquitetura da MC

Módulo	Elementos Lógicos	Freqüência (MHz)
Preditor de Vetores de Movimento	6,558	110.1
Acesso à Memória	1,307	153.2
Processador de Amostras	5,761	126.1
Compensador de Movimento	**13,892**	**99.2**

Dispositivo 2VP30FF1152-7

Uma versão *standard-cells* da arquitetura desenvolvida para a compensação de movimento foi gerada para permitir comparações com trabalhos relacionados publicados na literatura. Esta síntese foi realizada com a ferramenta Leonardo Spectrum usando uma tecnologia 0,18µm. Os resultados da versão *standard-cells* indicaram que a arquitetura utilizou 114.780 *gates* e atingiu uma freqüência de operação de 115,9 MHZ. Com esta freqüência, a arquitetura da MC é capaz de decodificar até 42,5 quadros totalmente bi-preditivos (B) por segundo ou até 74,5 quadros preditivos (P) por segundo.

Alguns trabalhos publicados na literatura foram usados como referência para melhor avaliar os resultados obtidos com a arquitetura da compensação de movimento. O trabalho desenvolvido por Wang (2005a) apresenta estratégias para reduzir a largura de banda de memória necessária para a MC. Os resultados apresentados em (WANG, 2005a) consideram somente o número de ciclos de *clock* usados para acessar a memória.

Além disso, é proposta uma hierarquia de memória capaz de reduzir, em média, 62% dos ciclos de acesso à memória. A arquitetura de MC apresentada nesta tese, considerando um ciclo de penalidade por troca de linha da memória, reduz em 80%, em média, o número de ciclos usados para acessar a memória.

Wang (2005) apresenta uma arquitetura para a MC do perfil *baseline* do padrão H.264/AVC. O artigo descreve o preditor de vetores de movimento e os interpoladores para posições fracionárias de pixel. O artigo não apresenta resultados de síntese para a MC completa e foca somente no interpolador de amostras de luminância. A arquitetura, operando a 100MHz, é capaz de decodificar quadros HDTV (não bi-preditivos) em tempo real. A arquitetura descrita nesta tese usa a idéia arquitetural do interpolador de amostras de luminância apresentado em (WANG, 2005), mas foram realizadas adaptações para permitir o suporte à bi-predição. Com estas modificações, a arquitetura dobra a taxa de processamento obtida com a arquitetura apresentada em (WANG, 2005).

O trabalho de Chen (2006) apresenta uma arquitetura combinada para predição inter-quadros e intra-quadro. Nesta solução, o preditor de vetores de movimento é implementado em software. A predição inter-quadros atinge uma redução de 48% na largura de banda de memória, enquanto que a arquitetura descrita nesta tese atinge uma redução de 64%. Além disso, a arquitetura desenvolvida nesta tese também reduz o número de ciclos usados no acesso à memória, como apresentado na Tabela 5.14. O número de ciclos de *clock* usados para acessar a memória pode ser muito importante quando pequenas áreas do quadro de referência são acessadas da memória, mas esta questão não é considerada nos trabalhos (WANG, 2005a) e (CHEN, 2006).

A Tabela 5.14 compara os resultados da arquitetura desenvolvida nesta tese com as soluções apresentadas em (WANG, 2005a) e (CHEN, 2006). Como pode ser observado na Tabela 5.14, a solução apresentada neste trabalho é a única que desenvolveu o preditor de vetores de movimento em hardware. O preditor de vetores consome uma grande quantidade de recursos de hardware, como pode ser observado na Tabela 5.14, onde o preditor de vetores consome quase metade dos recursos de hardware usados para a arquitetura completa da MC. Por isso, esta solução utiliza muito mais *gates* do que a solução proposta em (CHEN, 2006), onde o preditor de vetores foi implementado em software. A Tabela 5.14 também mostra que a arquitetura apresentada nesta tese atinge a maior redução nos acessos à memória dentre as três soluções. A arquitetura também reduz o tempo de execução da interpolação quando comparado com as outras soluções. É importante notar que enquanto (CHEN, 2006) apresenta seus resultados considerando o número médio de ciclos usados para decodificar um macrobloco, o resultado da arquitetura apresentada nesta tese considera o pior caso.

Os resultados obtidos para a arquitetura da compensação de movimento foram publicados em alguns eventos. O processador de amostras foi apresentado em (AZEVEDO, 2005), a solução para o acesso à memória foi publicada em (ZATT, 2007) e os resultados gerais da arquitetura foram apresentados em (AZEVEDO, 2007). Estes resultados foram considerados interessantes, porque esta foi a primeira solução publicada na literatura para a MC do perfil *main* do padrão H.264/AVC que foi completamente desenvolvida em hardware. Além disso, os resultados obtidos indicam que esta solução é capaz de decodificar vídeos de elevada resolução em tempo real.

Tabela 5.14: Comparação com outras arquiteturas de MC

	Wang (2005a)	**Chen (2006)**	**MC Proposta**
Predição de Vetores de Movimento	Incompleto	Software	Hardware
Redução na Largura de Banda de Memória	n/a	48%	62%
Tempo de Execução da Interpolação	560 ciclos/MB (pior caso)	320 ciclos/MB (média)	304 ciclos/MB (pior caso)
Número Total de Portas do MC	43K	40K	114K
Freqüência Necessária para Processar HD 1080	100MHz	87MHz	82MHz

6 CONCLUSÕES

"Eu vou voltar pra querência
Lugar onde fui parido"

João da Cunha Vargas

Esta tese apresentou soluções arquiteturais para módulos de codificadores e decodificadores de vídeo de acordo com o mais novo padrão da ITU-T e da ISO/IEC, o padrão H.264/AVC. O foco deste desenvolvimento arquitetural foram os módulos da estimação de movimento, da compensação de movimento, das transformadas diretas e inversas e da quantização direta e inversa.

Foram apresentados alguns conceitos básicos de compressão de vídeo e uma introdução ao padrão H.264/AVC. Nesta introdução, foram descritos os principais módulos de um codificador e de um decodificador H.264/AVC, de onde foi possível ter uma idéia mais concreta a respeito das operações realizadas em cada módulo. Além disso, foi apresentada uma análise de complexidade a respeito dos módulos formadores de codecs H.264/AVC.

Então, as arquiteturas desenvolvidas para os módulos das transformadas diretas e inversas, da quantização direta e inversa, da estimação de movimento e da compensação de movimento foram apresentadas. Além das arquiteturas para estes módulos, foi desenvolvida uma exploração no espaço de projeto para as transformadas e foi desenvolvida uma arquitetura que realiza os cálculos relativos a quaisquer das transformadas do padrão H.264/AVC e onde o paralelismo do processamento pode ser programado. Também foi desenvolvida uma investigação e comparação algorítmica para a estimação de movimento, incluindo diferentes algoritmos atuais e diversos critérios de comparação. Esta investigação teve o objetivo de embasar o desenvolvimento das arquiteturas desenvolvidas para este módulo. Foram três arquiteturas desenvolvidas para a estimação de movimento e estas soluções arquiteturais foram apresentadas no texto. Todas as arquiteturas desenvolvidas foram descritas em VHDL e foram mapeadas para FPGAs da Xilinx. Alguns dos módulos foram, também, sintetizados para *standard-cells*. Os resultados obtidos através da síntese destas arquiteturas foram apresentados e discutidos.

No decorrer do texto também foram referenciadas algumas das principais publicações geradas a partir das soluções arquiteturais desenvolvidas nesta tese.

Os resultados obtidos para os diversos módulos abordados na tese, bem como os trabalhos paralelos desenvolvidos em no grupo de pesquisa em TV Digital da UFRGS, mostraram-se à altura do desafio indicado na proposta desta tese. Todas as soluções

arquiteturais desenvolvidas são capazes de atingir tempo real para vídeos de elevada resolução (SDTV ou HDTV), mesmo quando a tecnologia alvo são FPGAs da família Virtex II Pro. As soluções para as transformadas e quantização direta e inversa atingiram com folga a taxa de processamento mínima de 93,3 milhões de amostras por segundo necessária para atingir tempo real (30 quadros por segundo) quando processando vídeos HDTV com 1920x1080 pixels. A solução paralela do módulo das transformadas diretas e a arquitetura multitransformada são capazes de atingir taxas de processamento superiores a três bilhões de amostras por segundo. Por outro lado, das soluções para a estimação de movimento, apenas a que usa o algoritmo *Full Search* com área de pesquisa 32x32 é capaz de processar vídeos com resolução HDTV em tempo real. As soluções com área de pesquisa com 64x64 amostras (*Full Search com Pel Subsampling* 4:1 e *Full Search com Pel Subsampling* 4:1 e *Block Subsampling* 4:1), em função das simplificações intencionalmente desenvolvidas, são capazes de atingir tempo real ao processar vídeos SDTV com 720x480 pixels.

Os módulos de hardware desenvolvidos em trabalhos de outros membros deste grupo de pesquisa para a predição intra-quadro e para o filtro redutor de efeito de bloco também apresentam desempenho para processar vídeos HDTV em tempo real, corroborando com a tese de que é viável desenvolver soluções em hardware na tecnologia atual para os módulos do padrão H.264/AVC capazes de atingir tempo real ao processar vídeos de elevada resolução.

Alguns resultados preliminares de desenvolvimento para o módulo da codificação de entropia também apontam para viabilidade de atingir tempo real para vídeos HDTV, embora alguns pontos ainda estejam em aberto neste módulo, em especial, no codificador.

As taxas de processamento obtidas em todas as soluções desenvolvidas indicam que é possível atingir tempo real para vídeos de elevada resolução, mesmo quando a tecnologia alvo são FPGAs Virtex II Pro da Xilinx. Sabidamente, para uma mesma tecnologia de fabricação, uma arquitetura direcionada para FPGAs apresenta um desempenho bastante inferior à mesma arquitetura quando direcionada diretamente para o silício, usando *standard-cells*, por exemplo. Além disso, o FPGA alvo para as soluções desenvolvidas nesta tese não é um dispositivo de última geração e, portanto, dispositivos mais novos são capazes de atingir uma freqüência de operação superior e, por conseqüência, as arquiteturas mapeadas para estes FPGAs atingiriam uma taxa de processamento mais elevada. Então, o uso de uma versão *standard-cells* ou de um FPGA mais atual, por certo, ampliariam ainda mais as taxas de processamento obtidas neste trabalho, reforçando a tese de que é viável atingir tempo real para os módulos do H.264/AVC na tecnologia atual.

Avaliando a integração dos módulos para formar um codificador ou decodificador H.264/AVC, ai a questão passa a ser um tanto mais complexa. Do lado do decodificador, é possível prever que será possível atingir tempo real para vídeos de alta definição quando o perfil *baseline* é considerado e condicionando esta previsão à adoção de um dispositivo da família dos FPGAs usados nesta tese, mas com mais recursos de hardware disponíveis. Com um dispositivo suficientemente grande (VP70 ou maior) será possível integrar todos os módulos desenvolvidos nesta tese e em trabalhos paralelos, evitando possíveis gargalos de roteamento. Se o perfil *main* for considerado, então o módulo do CABAC, ainda não completamente desenvolvido, pode causar algumas dificuldades, pois, como já mencionado neste texto, o comportamento estritamente bit-serial dos algoritmos utilizados neste módulo pode inviabilizar uma

solução com o paralelismo necessário para processar tempo real em vídeos HDTV. Mesmo com esta incerteza, os resultados preliminares apontam para a viabilidade de construções de arquiteturas para o CABAC capazes de atingir estes requisitos de processamento, mas com um elevado custo de hardware, devido às dificuldades para explorar o paralelismo necessário.

O lado do codificador apresenta desafios muito maiores, como já foi discutido neste texto. Em função das diversas opções do controle do codificador (modo de decisão), como escolha entre codificação inter-quadros ou intra-quadro, escolha do melhor tamanho de bloco na predição inter-quadros, escolha do melhor modo de operação na predição intra-quadro, entre outros. Para tomar esta decisão, o controle deve, idealmente, codificar todas as possibilidades de decisão e, na saída do codificador de entropia, para um mesmo parâmetro de quantização, avaliar qual é a melhor relação entre a taxa de compressão e a distorção causada. Assim, é possível abstrair este comportamento do codificador em um conjunto determinado de codificadores independentes operando para gerar os dados necessários para o controle tomar a decisão. Cada codificador independente seria responsável por tratar uma das opções do controle. Com os resultados obtidos até agora, é possível prever a viabilidade do desenvolvimento da arquitetura para um destes codificadores com desempenho necessário para atingir os requisitos de processamento. Então, é possível afirmar que, com o desenvolvimento de arquiteturas para os codificadores que operem em paralelo e de forma integrada ao controle, seria possível atingir tempo real para vídeos de alta definição. O problema desta solução está na quantidade abusivamente elevada de recursos de hardware necessários para o seu desenvolvimento. As estimativas iniciais de área indicam que seriam necessários diversos FPGAs trabalhando em paralelo para viabilizá-la, com um FPGA para cada codificador e um FPGA adicional para realizar o controle.

A integração do decodificador para o perfil *main* do padrão H.264/AVC, atualmente em desenvolvimento, será foco de investigações e desenvolvimentos próximos relacionados a esta tese. Para tanto, a solução para a decodificação de entropia será finalizada, incluindo o CABAC, o CAVLC e o Exp-Golomb. Atualmente a integração conta com os módulos das transformadas e quantização inversas, com a predição intra-quadro e com o filtro. A predição inter-quadros está sendo integrada enquanto esta tese está sendo escrita.

Cinco linhas de investigação e desenvolvimento formam os principais trabalhos futuros relacionados a esta tese. O primeiro trabalho, que já está em andamento, é a extensão da investigação algorítmica para a estimação de movimento, considerando blocos com tamanhos variáveis, bem como as demais ferramentas de codificação previstas pelo padrão H.264/AVC para este módulo.

Ainda considerando a estimação de movimento, mas sob o ponto de vista de hardware, estão sendo desenvolvidas duas novas arquiteturas para este módulo, uma utilizando o algoritmo *Diamond Search* com *Pel Subsampling* 2:1 e outra utilizando o algoritmo *Full Search*, mas considerando múltiplos tamanhos de blocos.

A terceira linha diz respeito à geração de uma solução completa e funcional para a codificação segundo o perfil *main*. Esta atividade, como já mencionado, apresenta ainda diversos desafios a serem atacados. Alguns módulos estão prontos e validados e outros estão em construção, como: a estimação e a compensação de movimento e a codificação

de entropia. Mas as principais questões são a integração e o problema da tomada de decisão pelo controle.

A quarta linha de investigações, que já teve início, será a adaptação das soluções desenvolvidas para o perfil *high* do padrão H.264/AVC. Atualmente, a compensação de movimento do decodificador e as transformadas diretas do codificador estão em processo de desenvolvimento.

Finalmente, a quinta linha de investigações e desenvolvimentos futuros diz respeito ao desenvolvimento arquitetural para codecs H.264/AVC escaláveis. Esta linha de ainda não teve início.

REFERÊNCIAS

AGOSTINI, L. V. et al. Forward and Inverse 2-D DCT Architectures Targeting HDTV for H.264/AVC Video Compression Standard. **Latin American Applied Research Journal**, [S.l.], v. 37, n. 1, p. 11-16, 2007.

AGOSTINI, L. V. et al. Design and FPGA Prototyping of a H.264/AVC Main Profile Decoder for HDTV. **Journal of Brazilian Computer Society**, [S.l.], v. 13, n. 1, p. 25-36, 2007a.

AGOSTINI, L. V. et al. High Throughput Multitransform and Multiparallelism IP Directed to the H.264/AVC Video Compression Standard. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2006. **Proceedings…** [S.l.]: IEEE, 2006. p. 5419-5422.

AGOSTINI, L. V. et al. FPGA Design of a H.264/AVC Main Profile Decoder for HDTV. In: IEEE INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, FPL, 2006. **Proceedings…** [S.l.]: IEEE, 2006a. p. 501-506.

AGOSTINI, L. V. et al. High Throughput Architecture for H.264/AVC Forward Transforms Block. In: ACM GRATE LAKES SYMPOSIUM ON VLSI, GLSVLSI, 2006. **Proceedings…** New York: ACM, 2006b. p. 320-323.

AGOSTINI, L. V. et al. High Throughput FPGA Based Architecture for H.264/AVC Inverse Transforms and Quantization. In: IEEE MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS, MWSCAS, 2006. **Proceedings…** [S.l.]: IEEE, 2006c.

AZEVEDO, A. P.; ZATT, B.; AGOSTINI, L. V.; BAMPI, S. MoCHA: a Bi-Predictive Motion Compensation Hardware for H.264/AVC Decoder Targeting HDTV. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2007. **Proceedings…** [S.l.]: IEEE, 2007.

AZEVEDO, A. P. **MoCHA:** Arquitetura Dedicada para a Compensação de Movimento em Decodificadores de Vídeo de Alta Definição, Seguindo o Padrão H.264**.** 2006. 120 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

AZEVEDO, A. et al. Motion Compensation Sample Processing for HDTV H.264/AVC Decoder. In: NORCHIP CONFERENCE, 2005. **Proceedings…** Piscataway: IEEE, 2005.

BANH, X.; TAN, Y. Adaptive Dual-Cross Search Algorithm for Block-Matching Motion Estimation. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 50, n. 2, p. 766-775, May 2004.

BHASKARAN, V.; KONSTANTINIDES, K. **Image and Video Compression Standards:** Algorithms and Architectures. 2nd ed. Boston: Kluwer Academic Publishers, 1997.

CHEN, J. et al. Low Complexity Architecture Design of H.264 Predictive Pixel Compensator for HDTV Applications, In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, ICASSP, 2006. **Proceedings…** [S.l.]: IEEE, 2006. v. 3, p. 932-935.

CHEN, K. et al. An Efficient Direct 2-D Transform Coding IP Design for MPEG-4 AVC/H.264/AVC. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings...** [S.l.]: IEEE, 2005. p. 4517-4520.

CHENG, Z. et al. High Throughput 2-D Transform Architectures for H.264/AVC Advanced Video Coders. In: IEEE ASIA-PACIFIC CONFERENCE ON CIRCUITS AND SYSTEMS, APCCAS, 2004. **Proceedings…** [S.l.]: IEEE, 2004. v. 2, p. 1141-1144.

CHIN, H. et al. A Bandwidth Efficient Subsampling-Based Block Matching Architecture for Motion Estimation. In: IEEE ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASPDAC, 2005. **Proceedings…** [S.l.]: 2005. v. 2, p. D7-D8.

CHU, C. et al. Hierarchical Global Motion Estimation/Compensation in Low Bitrate Video Coding. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1997. **Proceedings…** New York: IEEE, 1997. p. 1149-1152.

DIGILENT INC. **Digilent**. Disponível em: <www.digilentinc.com>. Acesso em: abr. 2007.

GHANBARI, M. **Standard Codecs:** Image Compression to Advanced Video Coding. United Kingdom: The Institution of Electrical Engineers, 2003.

GONZALEZ, R.; WOODS, R. **Processamento de Imagens Digitais**. São Paulo: E. Blücher, 2003.

HOROWITZ, M. et al, H.264/AVC Baseline Profile Decoder Complexity Analysis. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 704-716, July 2003.

HUANG, Y. et al. Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intraframe Frame Coder. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 15, n. 3, p. 378-401, Mar. 2005.

HUANG, Y. et al. An efficient and Low Power Architecture Design for Motion Estimation Using Global Elimination Algorithm. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, ICASSP, 2002. **Proceedings…** [S.l.]: IEEE, 2002. v. 3, p. 3120-3123.

INTERNATIONAL Organization for Standardization. ISO - International Organization for Standardization. Disponível em: <http://www.iso.org>. Acesso em: maio 2007.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 14496-2 - MPEG-4 Part 2 (01/1999)**: coding of audio visual objects – part 2: visual. [S.l.], 1999.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 11172 - MPEG-1 (11/1993)**: coding of moving pictures and associated audio for digital storage media up to about 1.5Mbit/s – part 2: video. [S.l.], 1993.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Home**. Disponível em: <www.itu.int/ITU-T/>. Acesso em: maio 2007.

INTERNATIONAL Telecommunication Union. Joint Video Team (JVT). Disponível em: <http://www.itu.int/ITU-T/studygroups/com16/jvt/>. Acesso em: maio 2007a.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264/AVC (03/05):** advanced video coding for generic audiovisual services. [S.l.], 2005.

INTERNATIONAL TELECOMMUNICATION UNION. **JVT-L050:** H.264/AVC AVC Fidelity Range Extensions. [S.l.], 2004.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264/AVC (05/03):** advanced video coding for generic audiovisual services. [S.l.], 2003.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.263 v3 (11/00):** video coding for low bit rate communication. [S.l.], 2000.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.262 (11/94):** generic coding of moving pictures and associated audio information – part 2: video. [S.l.], 1994.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.261 v1 (11/90):** video codec for audiovisual services at px64 kbit/s. [S.l.], 1990.

JAIN, J.; JAIN, A. Displacement Measurement and Its Application in Interframe Image Coding. **IEEE Transactions on Communications**, [S.l.], v. 29, n. 12, p. 1799-1808, Dec. 1981.

JING, X.; CHAU, L. An Efficient Three-Step Search Algorithm for Block Motion Estimation. **IEEE Transactions on Multimedia**, [S.l.], v. 6, n. 3, p. 435-438, June 2004.

JOINT VIDEO TEAM (JVT). **Reference Software**. Disponível em: <http://ftp3.itu.ch/av-arch/jvt-site/reference_software/>. Acesso em: maio 2007.

KAMACI, N.; ALTUNBASAK, Y. Performance Comparison of the Emerging H.264/AVC Video Coding Standard with the Existing Standards. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO, ICME, 2003. **Proceedings...** [S.l.]: IEEE, 2003. v. 1, p. I345-I348.

KANNANGARA, C.; RICHARDSON, I. Computational Control of an H.264/AVC Encoder through Lagrangian Cost Function Estimation. In: INTERNATIONAL WORKSHOP ON VERY LOW BITRATE VIDEO, 2005. **Proceedings…** [S.l.:s.n.], 2005.

KORAH, R. et al. Motion Estimation with Candidate Block and Pixel Subsampling Algorithm. In: IEEE INTERNATIONAL WORKSHOP ON IMAGING SYSTEMS AND TECHNIQUES, IST, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 130-133.

KORDASIEWICZ, R.; SHIRANI, S. Hardware Implementation of the Optimized Transform and Quantization Blocks of H.264/AVC. In: CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING. **Proceedings…** [S.l.]: IEEE, 2004. v. 2, p. 943-946.

KROUPIS, N. et al. A Modified Spiral Search Motion Estimation Algorithm and its Embedded System Implementation. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 3347-3350.

KUHN, P. **Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation**. Boston: Kluwer Academic Publishers, 1999.

KWON, S. et al. Overview of H.264/AVC / MPEG-4 Part 10. In: INTERNATIONAL CONFERENCE ON MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES, WSEAS, 2005. **Proceedings…** [S.l.:s.n.], 2005.

LEE, K. et al. QME: An Efficient Subsampling-Based Block Matching Algorithm for Motion Estimation. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2004. **Proceedings…** [S.l.]: IEEE, 2004. v. 2, p. 305-308.

LI, R. et al. A New Three-Step Search Algorithm for Block Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology,** [S.l.], v. 4, n. 4, p. 438-442, Aug. 1994.

LIN, H. et al. Combined 2-D Transform and Quantization Architectures for H.264/AVC Video Coders. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings...** [S.l.]: IEEE, 2005. p. 1802-1805.

LIN, C.; LEOU, J. An Adaptative Fast Full Search Motion Estimation Algorithm for H.264/AVC. In: IEEE INTERNATIONAL SYMPOSIUM CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005a. p. 1493-1496.

LOUKIL, H. et al. Hardware Implementation of Block Matching Algorithm with FPGA Technology. In: INTERNATIONAL CONFERENCE ON MICROELECTRONICS, ICM, 2004. **Proceedings…** [S.l.:s.n.], 2004. p. 542-546.

MALVAR, H. et al. Low-Complexity Transform and Quantization in H.264/AVC. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 598-603, July 2003.

MARPE, D. et al. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 620-636, July 2003.

MENTOR GRAPHICS CORPORATION. **ModelSim SE User's Manual - Software Version 6.2d**. [S.l.], 2005. Disponível em: <www.model.com/support>. Acesso em: set. 2005.

MOHAMMADZADEH, M.; ESHGHI, M.; AZADFAR, M. Parameterizable Implementation of Full Search Block Matching Algorithm Using FPGA for Real-Time Applications. In: INTERNATIONAL CARACAS CONFERENCE ON DEVICES, CIRCUITS AND SYSTEMS, ICCDCS, 2004. **Proceedings…** [S.l.:s.n.], 2004. p. 200–203.

POLLACK, J. Displays of a Different Stripe. **IEEE Spectrum**, [S.l.], v. 43, n. 8, p. 40-44, Aug. 2006.

PORTO, R.; PORTO, M.; BAMPI, S,; AGOSTINI, L. High Throughput Architecture for Forward Transforms Module of H.264/AVC Video Coding Standard. IN: IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS, ICECS, 2007. **Proceedings…** [S.l.]: IEEE, 2007.

PORTO, M.; AGOSTINI, L.; BAMPI, S. Investigação de Algoritmos e Desenvolvimento Arquitetural para a Estimação de Movimento em Compressão de Vídeo Digital. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, CLEI, 2006. **Proceedings…** Chile: Universidad de Santiago de Chile, 2006. 1 CDROM.

PORTO, M. et al. Design Space Exploration on the H.264/AVC 4x4 Hadamard Transform. In: NORCHIP CONFERENCE, 2005. **Proceedings…** Piscataway: IEEE, 2005.

PURI, A. et al. Video Coding Using the H.264/MPEG-4 AVC Compression Standard. **Elsevier Signal Processing: Image Communication,** [S.l.], n. 19, p.793–849, 2004.

RICHARDSON, I. **H.264/AVC and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia**. Chichester: John Wiley and Sons, 2003.

RICHARDSON, I. **Video Codec Design – Developing Image and Video Compression Systems**. Chichester: John Wiley and Sons, 2002.

ROMA, N.; DIAS, T.; SOUSA, L. **Customisable Core-Based Architectures for Real-Time Motion Estimation on FPGAs**. Berlin: Springer-Verlag, 2003. p. 745-754. (Lecture Notes in Computer Science, v. 2778).

ROSA, V. S.; STAEHLER, W. T.; AZEVEDO, A. P.; ZATT, B.; PORTO, R. E.; AGOSTINI, L. V.; BAMPI, S.; SUSIN, A. FPGA Prototyping Strategy for a H.264/AVC Video Decoder. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING, RSP, 2007. **Proceedings…** [S.l.]: IEEE, 2007. p. 174-180.

SAHAFI, L. **Context-Based Complexity Reduction Applied to H.264/AVC Video Compression**. 2005. 70 f. Thesis (Master in Applied Science) – School of Engineering Science, Simon Frase University, Canada.

SALOMON, D. **Data Compression:** The Complete Reference. 2nd ed. New York: Springer, 2000.

SANBORN, S.; MA, X. Quantifying Information Content in Data Compression Using the Autocorrelation Function. **IEEE Signal Processing Letters**, [S.l.], v.12, p. 230-233, Mar. 2005.

SHI, Y.; SUN, H. **Image and Video Compression for Multimedia Engineering:** Fundamentals, Algorithms and Standards**.** Boca Raton: CRC Press, 1999.

SULLIVAN, G.; WIEGAND, T. Video Compression – From Concepts to the H.264/AVC Standard. **Proceedings of the IEEE**, [S.l.], v. 93, n. 1, p. 18-31, Jan. 2005.

SULLIVAN, G. et al. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In: CONFERENCE ON APPLICATIONS OF DIGITAL IMAGE PROCESSING, SPIE, 2004. **Proceedings…** Denver: SPIE, 2004.

SULLIVAN, G.; WIEGAND, T. Rate-Distortion Optimization for Video Compression. **IEEE Signal Processing Magazine**, [S.l.], v. 15, p. 74-90, Nov. 1988.

SUNNA, P. AVC / H.264/AVC – An Advanced Video Coding System for SD and HD Broadcasting. **European Broadcasting Union Technical Review**, [S.l.], n. 302, Apr. 2005. Disponível em: <http://www.ebu.ch/departments/technical/trev/trev_302-sunna.pdf>. Acesso em: set. 2005.

SYNPLICITY INC. **Synplicity**: Home. Disponível em: <www.synplicity.com>. Acesso em: maio 2007.

TOURAPIS, M. et al. Fast Motion Estimation Using Circular Zonal Search. In: VISUAL COMMUNICATIONS AND IMAGE PROCESSING. **Proceedings…** San Jose: [s.n.], 1999. v. 2, p. 1496-1504.

VQEG: The Video Quality Experts Group Web Site. Disponível em: <www.its.bldrdoc.gov/vqeg/>. Acesso em: abr. 2007.

WANG, S. et al. A New Motion Compensation Design for H.264/AVC Decoder. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 4558-4561.

WANG, R. LI, J.; HUANG, C. Motion Compensation Memory Accesss Optimization Strategies for h.264/AVC Decoder. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, ICASSP, 2005. **Proceedings…** [S.l.]: IEEE, 2005a. v. 5, p. 97-100.

WANG, T. et al. Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264/AVC. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2003. **Proceedings...** [S.l.]: IEEE, 2003. p. II800-II803.

WIEGAND, T. et al. Rate-Constrained Coder Control and Comparison of Video Coding Standards. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 688-703, July 2003.

WIEGAND, T. et al. Overview of the H.264/AVC Video Coding Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 560-576, July 2003a.

WIEN, M. Variable Block-Size Transforms for H.264/AVC. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 604-613, July 2003.

WU, D. et al. A Single Scalar DSP based Programmable H.264/AVC Decoder. In: SWEDISH SYSTEM-ON-CHIP CONFERENCE, SSoCC, 2005. **Proceedings…** [S.l.:s.n.], 2005.

XILINX INC. **Xilinx**: The Programmable Logic Company. Disponível em: <www.xilinx.com>. Acesso em: abr. 2007.

XILINX INC. **Xilinx ISE 8.2i Software Manuals and Help.** [S.l.], 2006. Disponível em: <www.xilinx.com/support/sw_manuals/xilinx82/index.htm>. Acesso em: dez. 2006.

XILINX INC. **Xilinx University Program Virtex-II Pro Development System - Hardware Reference Manual**. [S.l.], 2006. Disponível em: <http://www.xilinx.com>. Acesso em: dez. 2006a.

XILINX INC. **Embedded System Tools Reference Manual - Embedded Development Kit, EDK 8.2i**. [S.l.], 2006. Disponível em: <http://www.xilinx.com>. Acesso em: dez. 2006b.

XILINX INC. **Virtex-II Pro and Virtex-II Pro X Platform FPGAs:** Complete Data Sheet. [S.l.], 2005. Disponível em: <www.xilinx.com>. Acesso em: set. 2005.

YI, X.; LING, N**.** Rapid Block-Matching Motion Estimation Using Modified Diamond Search Algorithm. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 5489-5492.

ZATT, B.; AZEVEDO, A. P.; AGOSTINI, L. V.; SUSIN, A.; BAMPI, S. Memory Hierarchy Targeting Bi-Predictive Motion Compensation for H.264/AVC Decoder. In: IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI, ISVLSI, 2007. **Proceedings…** Los Alamitos: IEEE, 2007. p. 445-446.

ZHANG, J. et al. Performance and Complexity Joint Optimization for H.264/AVC Video Coding. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2003. **Proceedings…** [S.l.]: IEEE, 2003. v. 2, p. II888-II891.

ZHU, C; LIN, X; CHAU, L. Hexagon-Based Search Pattern for Fast Block Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 12, n. 5, p. 349-355, May 2002.

APÊNDICE A

DESCRIÇÃO DA METODOLOGIA DE VALIDAÇÃO E PROTOTIPAÇÃO DAS ARQUITETURAS DESENVOLVIDAS

Este apêndice descreve, resumidamente, o processo de validação e prototipação das arquiteturas desenvolvidas nesta tese.

A.1 Metodologia de Validação das Arquiteturas

Uma abordagem de simulação exaustiva foi escolhida para validar as arquiteturas desenvolvidas. Esta abordagem requer um grande número de casos de teste, conduzindo a um elevado tempo de execução das simulações.

Os dados de entrada para a validação foram extraídos diretamente do software de referência do padrão H.264/AVC (JVT, 2007). Para tanto, funções foram inseridas no software de referência para capturar os dados de interesse para a validação. Então, seqüências de vídeos reais foram processadas pelo software de referência com as funções adicionais de captura de dados para validação. Estas funções capturaram os dados de cada módulo e salvaram os dados de forma organizada em arquivos texto.

A ferramenta ModelSim (MENTOR, 2005) da Mentor Graphics foi usada para rodar as simulações necessárias para a validação de todos os módulos desenvolvidos. *Testbenches* foram descritos em VHDL para gerenciar as entradas e saídas das arquiteturas. Os estímulos de entrada foram extraídos dos arquivos gerados pelas funções inseridas no software de referência. O *testbench* organiza os resultados das saídas das arquiteturas e armazena estes resultados em novos arquivos texto. A comparação entre os resultados gerados pela arquitetura com os resultados gerados pelo software de referência indica se a arquitetura está validada ou não. Esta comparação foi automatizada, através de um programa descrito em C. Estes passos foram repetidos exaustivamente para algumas seqüências de vídeo.

O primeiro passo para a validação consiste na execução da simulação de um modelo comportamental das arquiteturas. O segundo passo repete o processo de validação, mas considerando um modelo pós *place-and-route* das arquiteturas. Para gerar o modelo pós *place-and-route*, a descrição VHDL foi sintetizada para o FPGA alvo. O resultado da síntese já considera a disposição física da arquitetura dentro do FPGA e inclui atrasos. Neste passo, a ferramenta ISE da Xilinx (XILINX, 2006) foi usada em conjunto com a ferramenta ModelSim para gerar o modelo pós *place-and-route* das arquiteturas. O

dispositivo alvo selecionado foi um FPGA XC2VP30 (XILINX, 2005) da família Virtex-II Pro da Xilinx.

Para validar os modelos desenvolvidos, as saídas do modelo pós *place-and-route* foram novamente comparadas com os resultados gerados pelo software de referência. Após pequenas correções no código VHDL, a comparação não encontrou nenhuma diferença entre os resultados das arquiteturas e os resultados gerados pelo software de referência. Então, as arquiteturas foram consideradas validadas.

A.2 Prototipação das Arquiteturas

Algumas das arquiteturas desenvolvidas foram prototipadas em FPGAs Xilinx, entre elas, as arquiteturas dos módulos das transformadas diretas serial, das transformadas diretas paralela, das transformadas inversas serial, da quantização inversa, da estimação de movimento com *Full Search* e da compensação de movimento.

A prototipação foi realizada usando a placa de desenvolvimento XUP-V2P (XILINX, 2006a), produzida pela Digilent Inc (DIGILENT, 2007). Esta placa, apresentada na Figura A.1, possui um FPGA XC2VP30 da família Virtex II Pro da Xilinx (com dois processadores PowerPC 405 embarcados), porta serial, saída VGA e várias outras interfaces e recursos. A placa também contém uma memória SDRAM com 512MB.

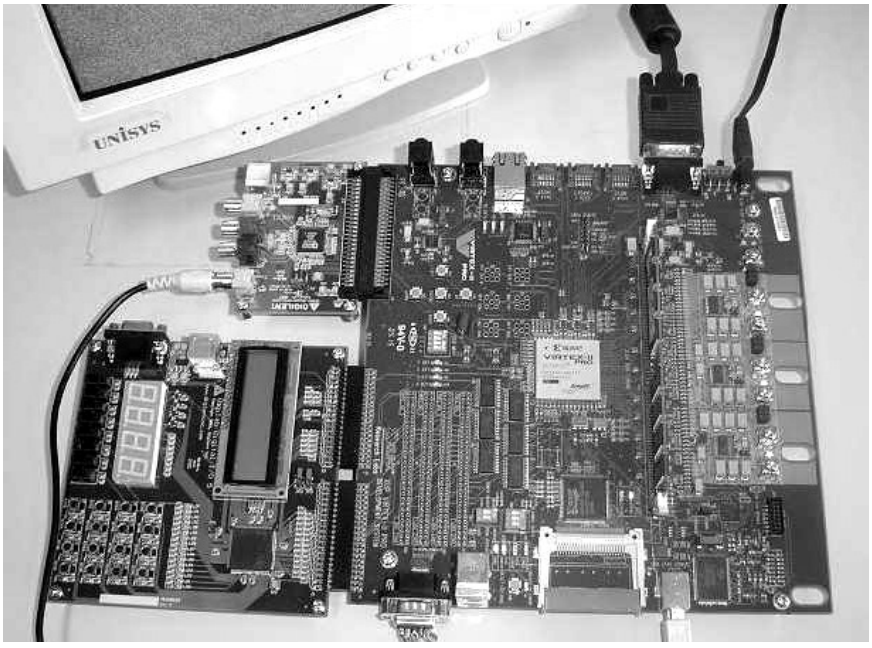

Figura A.1: Placa XUP-V2P da Digilent usada na prototipação

A ferramenta EDK da Xilinx (XILINX, 2006b) foi usada para criar a plataforma de programação, que é necessária para o processo de prototipação. Cada arquitetura desenvolvida foi conectada ao barramento do processador e foi prototipada individualmente. Um dos processadores PowerPC disponíveis foi usado como um controlador para a arquitetura prototipada, emulando os outros módulos de um codec. O estímulo de entrada foi enviado para o sistema de prototipação através da porta RS-232, usando um programa terminal rodando em um PC. O processador envia os estímulos para a arquitetura prototipada e os resultados gerados pela arquitetura são enviados novamente para o PC. Os resultados do módulo prototipado são comparados com os

resultados do software de referência, para verificar se a arquitetura prototipada está operando corretamente no FPGA. Esta comparação é realizada no PC, após o final dos cálculos do módulo.

Os módulos prototipados foram verificados funcionalmente. Um processador PowerPC foi sintetizado com o módulo prototipado em verificação funcional (MPVF) e todos os estímulos de entrada do MPVF, incluindo o sinal de clock, são gerados pelo processador e enviados para o MPVF, através do barramento do processador. As saídas do MPVF são capturadas diretamente do barramento. Como, nesta abordagem, o *clock* para o MPVF é gerado por software, este *clock* acaba sendo muito inferior ao *clock* para o qual a arquitetura prototipada está apta a operar. Ainda assim, com esta abordagem é possível verificar o correto funcionamento do módulo prototipado, mesmo quando este módulo está operando com uma freqüência menor do que a freqüência para qual ele foi projetado para operar. A Figura A.2 mostra esta abordagem de prototipação.

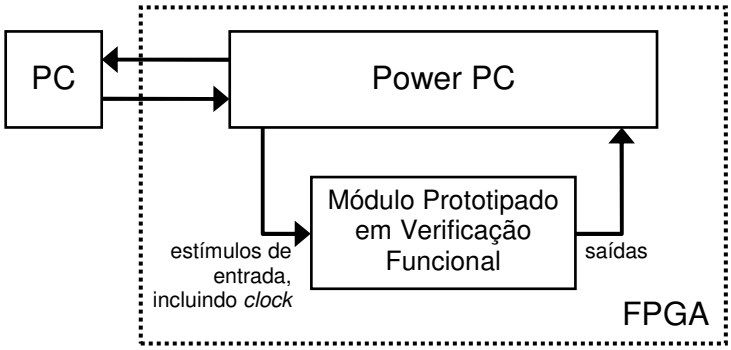

Figura A.2: Abordagem para verificação funcional dos protótipos

A verificação funcional foi bem sucedida e todos os módulos prototipados, após alguns pequenos ajustes em seus códigos VHDL, acabaram por operar corretamente.

Para o caso da prototipação do módulo da compensação de movimento, um *buffer* de quadros VGA foi sintetizado junto com a arquitetura e alguns resultados visuais foram enviados diretamente da placa de prototipação para um monitor. A Figura A.3 mostra um exemplo de saída de vídeo para o protótipo do compensador de movimento.

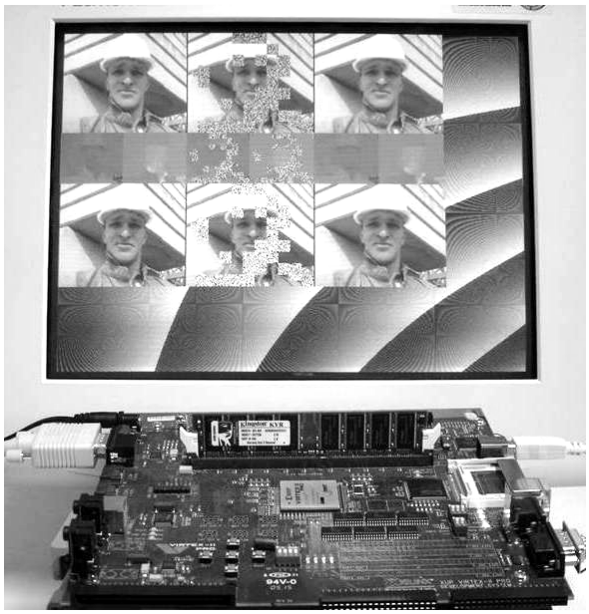

Figura A.3: Protótipo do módulo da compensação de movimento

APÊNDICE B

DESCRIÇÃO DOS ALGORITMOS DE BUSCA PARA ESTIMAÇÃO DE MOVIMENTO INVESTIGADOS

Os algoritmos de busca para a estimação de movimento determinam a forma como o melhor casamento (*best matching*), para o bloco do quadro atual, será buscado dentro da área de pesquisa do quadro de referência. O algoritmo de busca tem influência direta na complexidade computacional da estimação, bem como na qualidade dos vetores gerados.

As próximas seções apresentam uma breve descrição de cada um dos algoritmos investigados nesta tese: *Full Search*, *Three Step Search*, *One at a Time Search*, *Diamond Search*, *Hexagon Search* e *Dual Cross Search*. Além disso, são apresentadas duas técnicas de subamostragem que foram empregadas nas investigações algorítmicas apresentas nesta tese: *Pel Subsampling* e *Block Subsampling*.

B.1 *Full Search*

O algoritmo *Full Search* (FS) procura o melhor casamento para o bloco atual na área de pesquisa comparando este bloco com todos os blocos candidatos existentes dentro da área de pesquisa do quadro de referência (BHASKARAN, 1997; LIN, 2005a). Quando todos os blocos candidatos tiverem sido avaliados, então o bloco que apresenta a maior similaridade com o bloco atual é escolhido e, assim, será gerado um vetor de movimento referente ao deslocamento do bloco atual em relação ao bloco de maior similaridade na área de busca do quadro de referência. O algoritmo aplica a função de similaridade para todos os pixels de todos os blocos candidatos em relação ao bloco atual.

O algoritmo FS possui uma grande complexidade computacional. Como o algoritmo calcula a diferença para todas as posições possíveis na área de pesquisa, a sua complexidade é diretamente proporcional às dimensões da área de pesquisa.

B.2 *Three Step Search*

O *Three Step Search* (TSS) é mais difundido com o nome de busca em três passos. No entanto, ele pode ser estendido para *n* passos (JING, 2004). O principal objetivo deste algoritmo é reduzir drasticamente o número de comparações, se comparado ao algoritmo *Full Search*. Para isso, um número finito de comparações é determinado e dividido nos três passos do algoritmo.

Para uma área de pesquisa de +/- $(2^n - 1)$ pixels, o passo inicial S deve ser inicializado com $S = 2^{n-1}$. O primeiro passo consiste em posicionar a busca no centro da área de pesquisa e calcular o erro para esta posição. Em seguida, calcular mais oito valores +/- S pixels em torno da posição (0,0) (centro da área de pesquisa). Então são comparados os nove valores de erro obtidos e é determinada como nova posição de origem a posição de menor erro. No segundo passo a variável S é dividida por dois e novamente oito valores +/- S pixel em torno da origem são calculados. Desta vez, oito valores são comparados e, novamente, a origem será substituída pela posição com o menor erro. No terceiro e último passo, mais uma vez a variável S é dividida por dois. Desta vez ela irá conter o valor um, o que indica o último estágio de busca. Mais uma vez, os oito valores de erros são comparados e o vetor de movimento será gerado para a posição com o menor valor de erro. Em geral, $8n + 1$ comparações são necessárias para uma busca em uma área de pesquisa de +/- $(2n-1)$.

B.3 *One at a Time Search*

O algoritmo *One at a Time Search* (OT) é bastante simples e visa reduzir ainda mais o número de comparações. O algoritmo baseia-se na idéia de encontrar, primeiro, um mínimo horizontal e, em seguida, um mínimo vertical (RICHARDSON, 2002). O algoritmo começa calculando o erro para a posição central da área de pesquisa. Em seguida, mais dois valores, um imediatamente à direita e outro imediatamente à esquerda, são calculados. Caso o valor do centro seja o menor, o algoritmo passa para o estágio de busca vertical. Caso contrário, redefine-se o centro com a posição do menor erro e calcula-se o valor para o vizinho, seja ele à direita ou à esquerda (dependendo do valor escolhido no passo anterior). O estágio de cálculo dos valores verticais é muito semelhante, variando a busca para cima e para baixo. A busca termina quando o valor de SAD calculado é maior que o valor encontrado para o bloco anterior.

B.4 *Diamond Search*

O algoritmo *Diamond Search* (DS) possui dois padrões diamante, *Large Diamond Search Pattern* (LDSP) e *Small Diamond Search Pattern* (SDSP) que são usados na etapa inicial e final do algoritmo respectivamente (KUHN, 1999; YI, 2005). O padrão LDSP consiste em 9 comparações e é utilizado na etapa inicial da pesquisa. Já o padrão SDSP consiste em 4 comparações e é utilizado na etapa final da pesquisa, com o intuito de refinar o resultado obtido na etapa anterior (KUHN, 1999).

O algoritmo começa aplicando o padrão LDSP ao centro da área de pesquisa. Caso o valor de menor erro seja encontrado no centro, o algoritmo aplica o padrão SDSP para refinar o resultado obtido. Caso contrário, um novo padrão diamante é aplicado à posição de menor erro da etapa anterior. Esta posição pode pertencer a uma aresta ou a um vértice do diamante.

No caso da busca por uma aresta, mais 3 valores são calculados para formar um novo diamante em torno da nova origem. Quando o novo centro é um vértice do diamante, mais 4 valores são calculados para formar o novo diamante em torno do centro. Caso o menor erro não seja encontrado no centro, a etapa de busca será repetida, seja ela por uma aresta ou por um vértice. Quando o menor erro for encontrado para o centro do diamante o padrão SDSP é aplicado. Então, mais quatro valores imediatamente ao redor do centro serão calculados e a posição com o menor erro será a escolhida.

B.5 *Hexagon Search*

O algoritmo *Hexagon Search* (HS) (ZHU, 2002) pode ser considerado uma evolução do algoritmo *Diamond Search*. Assim como no algoritmo *Diamond Search*, o *Hexagon* possui dois padrões em hexágono, um para a etapa inicial *Large Hexagon Pattern*, com 7 cálculos, e outro para a etapa final, com 4 cálculos, *Small Hexagon Pattern*.

O algoritmo começa no centro da área de pesquisa aplicando o *Large Hexagon Pattern*. Caso o melhor casamento seja encontrado no centro do hexágono, o algoritmo aplica o *Small Hexagon Pattern*. Caso o melhor casamento seja encontrado em um dos vértices do hexágono, este vértice passa a ser o centro de outro hexágono, e mais 3 posições são calculadas. Este processo é repetido até que o melhor resultado seja obtido para o centro do hexágono. Então, o *Small Hexagon Pattern* é aplicado a este centro e mais 4 pontos são avaliados. O melhor resultado dentre esses 5 pontos será o escolhido. Pode-se perceber que o algoritmo *Hexagon* pode reduzir o número de comparações se comparado ao algoritmo *Diamond Search*. O *Hexagon* pode sair na primeira etapa, aplicando os padrões *Large* e *Small Hexagon*, com 11 comparações, contra 13 do algoritmo *Diamond*. A cada iteração do padrão *Large*, o algoritmo calcula três novos pontos, ao contrário do algoritmo *Diamond* que pode calcular três ou quatro pontos a cada iteração.

B.6 *Dual Cross Search*

O algoritmo *Dual Cross Search* (DCS) visa reduzir ainda mais o número de comparações realizadas pelos algoritmos *Diamond* e *Hexagon Search*. Este algoritmo também possui dois padrões de pesquisa, o 2x2 *Cross Search Pattern* e o 4x4 *Cross Search Pattern* (BANH, 2004). O algoritmo começa aplicando o padrão 2x2 ao centro da área de pesquisa. São calculados o centro e mais 4 pontos imediatamente ao redor. Os resultados são comparados e se o melhor resultado é encontrado no centro, a busca termina, caso contrário, o padrão 4x4 é aplicado, sendo seu centro o melhor resultado obtido no passo anterior. O padrão *Cross* 4x4 calcula mais três pontos a cada iteração e compara os novos valores com o centro do *Cross*. Este laço será repetido até que o melhor resultado seja encontrado no centro.

Para todos os casos onde o melhor resultado não é encontrado no *Cross* 2x2, um refinamento final é aplicado ao resultado obtido no *Cross* 4x4. Este algoritmo pode reduzir o número de comparações se comparado ao algoritmo *Hexagon*, pois a busca pode ser interrompida na primeira etapa, com quatro comparações, ao invés das 11 comparações (mínimo) necessárias para concluir uma busca com o algoritmo *Hexagon*. O DCS também calcula três novas posições a cada iteração no *Cross* 4x4, assim como o algoritmo *Hexagon*, no entanto, a etapa de refinamento do DCS calcula apenas mais três pontos, contra quatro do *Hexagon*.

B.7 Técnicas de Subamostragem

A investigação dos algoritmos de estimação de movimento considerou, além dos algoritmos apresentados acima, duas técnicas de subamostragem que foram associadas a estes algoritmos. A subamostragem a nível de blocos (*Block Subsampling*) pode ser aplicada apenas ao algoritmo *Full Search*, enquanto que a subamostragem a nível de pixel (*Pel Subsampling*) pode ser aplicada a todos os algoritmos de busca investigados neste trabalho.

B.7.1 *Pel Subsampling*

A técnica de *Pel Subsampling* (PS) pode ser agregada a qualquer algoritmo de estimação de movimento. A idéia é reduzir o número de comparações para cada bloco candidato, pois uma parcela dos pixels não é usada nos cálculos (KUHN, 1999). A cada comparação, o algoritmo calcula a diferença para alguns pixels do bloco e simplesmente descarta os outros. Isto diminui o tempo de processamento e a complexidade computacional do algoritmo. A técnica de *Pel Subsampling* pode ser aplicada em qualquer proporção, como 2:1, 4:1, 8:1, por exemplo.

Esta técnica reduz a complexidade computacional do algoritmo, no entanto, os vetores resultantes geram resultados mais elevados de erro do que as versões sem subamostragem. Os resultados tendem a piorar proporcionalmente ao aumento da subamostragem.

B.7.2 *Block Subsampling*

A técnica de *Block Subsampling* (BS) (KORAH, 2005) pode ser aplicada apenas ao algoritmo de *Full Search* e visa acelerar o processo de busca. Da mesma forma que a técnica de *Pel Subsampling*, uma subamostragem é realizada, mas não ao nível de pixel. O *Block Subsampling* utiliza a subamostragem ao nível de blocos. Nesta técnica, não são comparados todos os blocos candidatos da área de pesquisa, os blocos são subamostrados, sendo que a diferença é calculada para alguns e outros são simplesmente descartados. Assim como o PS, o BS pode ser aplicado com os mais diversos níveis de subamostragem, como 2:1, 4:1, 8:1 entre outros.

Esta técnica também reduz significativamente o número de comparações, se comparado ao algoritmo FS sem subamostragem. O algoritmo BS também pode ser aplicado em conjunto com a técnica *Pel Subsampling*. Desta forma, aplica-se uma subamostragem ao nível de pixel aos blocos que não foram excluídos no processo de subamostragem a nível de bloco. Isto acelera ainda mais o processo, no entanto, aumenta a probabilidade do bloco escolhido possuir o resultado bastante distante do ótimo em termos de erro.

Informática  *UFRGS*

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

"Desenvolvimento de Arquitetura de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC"

por

Luciano Volcan Agostini

Tese apresentada aos Senhores:

Prof. Dr. Altamiro Amadeu Susin

Prof. Dr. Ricardo Augusto da Luz Reis

Prof. Dr. Eduardo Antonio Barros da Silva (UFRJ)

Prof. Dr. Ricardo Pezzuol Jacobi (UnB)

Vista e permitida a impressão.
Porto Alegre, 13 /08/ 07

Prof. Dr. Sérgio Bampi,
Orientador.

Prof. Dr. Ivan Saraiva Silva (UFRN)
Co-orientador.

Profa. Luciana Porcher Nedel
Coordenadora do Programa de
Pós-Graduação em Computação - PPGC
Instituto de Informática - UFRGS