Universidade Federal de Pelotas

Trabalho de Conclusão de Curso

André Marcelo Coelho da Silva

DESENVOLVIMENTO DE UM COMPENSADOR DE MOVIMENTO PARA O PADRÃO H.264 DE COMPRESSÃO DE VÍDEO

Pelotas, 2006

André Marcelo Coelho da Silva

DESENVOLVIMENTO DE UM COMPENSADOR DE MOVIMENTO PARA O PADRÃO H.264 DE COMPRESSÃO DE VÍDEO

Trabalho acadêmico apresentado ao curso de Bacharelado em Ciência da Computação do Instituto de Física e Matemática, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Luciano Volcan Agostini

Co- Orientador: Prof. Dr. José Luís A. Güntzel

Pelotas, 2006

Dados de catalogação na fonte:
Ubirajara Buddin Cruz – CRB-10/901
Biblioteca de Ciência & Tecnologia - UFPel

S586d Silva, André Marcelo Coelho da
Desenvolvimento de um compensador de movimento para o padrão h.264 de compressão de vídeo / André Marcelo Coelho da Silva ; orientador Luciano Volcan Agostini ; co-orientador José Luís Almada Güntzel. – Pelotas, 2006. – 63f. : il. - Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. Instituto de Física e Matemática. Universidade Federal de Pelotas. Pelotas, 2006.

1.Informática. 2.Compressão de vídeo. 3.Compensação de movimento. 4.Padrão H.264. 5.VHDL. 6.FPGA. I.Agostini, Luciano Volcan. II.Güntzel, José Luís Almada. III.Título.

CDD: 006.62

DESENVOLVIMENTO DE UM COMPENSADOR DE MOVIMENTO PARA O PADRÃO H.264 DE COMPRESSÃO DE VÍDEO

André Marcelo Coelho da Silva

Monografia defendida e aprovada em Agosto de 2006, pela banca examinadora constituída pelos seguintes professores:

_____

Prof. MSc. Luciano Volcan Agostini – Orientador
Universidade Federal de Pelotas – UFPel

_____

Prof. Dr. Lucas Ferrari de Oliveira
Universidade Federal de Pelotas – UFPel

_____

Prof. MSc. Marcello da Rocha Macarthy
Universidade Federal de Pelotas – UFPel

_____

Prof. MSc. Sandro Silva
Centro Federal de Educação Tecnológica de Pelotas – CEFET-RS

Agradecimentos

Aos meus pais, Elmo Ribeiro da Silva e Gelsinete Coelho da Silva, pela educação que me concederam e por sempre me darem apoio e respaldo para todas as minhas ações acadêmicas e profissionais, e que foram, são e sempre serão os meus espelhos para todas as decisões que por mim forem tomadas.

A minha irmã, Andréia Coelho da Silva, por ser uma pessoal especial e uma amiga para os momentos mais difíceis e que está sempre disposta a me ajudar.

A minha avó, Nair Vargas Coelho, por ser a minha segunda mãe e por isso ter um papel fundamental na minha educação, e onde eu sempre encontrei uma palavra de incentivo e carinho.

Ao meu orientador e ao meu co-orientador, Luciano Agostini e José Luís Güntzel respectivamente, por serem excepcionais professores e orientadores, além de serem dois amigos que me fizeram crescer muito, pessoal e profissionalmente, nestes anos de convívio e que nunca mediram esforços para ajudar a mim e a todos do nosso grupo de pesquisa.

A todos os amigos e colegas pela convivência durante estes anos e por terem me ajudado de alguma forma na conclusão deste trabalho, porém um agradecimento especial a colega de GACI, Fabiane Konrad Rediess, e ao colega de classe e do GACI, Marcelo Schiavon Porto, pela suas contribuições ao longo deste trabalho.

Resumo

SILVA, André Marcelo C. da **Desenvolvimento de um Compensador de Movimento para o Padrão H.264 de Compressão de Vídeo.** 2006. 63f. Monografia – Curso de Bacharelado em Ciência da Computação, Universidade Federal de Pelotas, Pelotas.

Vídeos digitais estão demandando quantidades cada vez maiores de memória, o que dificulta bastante sua transferência via rede e o seu armazenamento e manipulação. Em função de tais problemas, têm sido desenvolvidos padrões de compressão de vídeo, tais como, MPEG-1, MPEG-2, H.263 e padrões mais novos, como, MPEG-4 e H.264, este último tendo seu rascunho final aprovado em outubro de 2003.

Uma importante operação utilizada nestes padrões de compressão de vídeos é a compensação de movimento, sendo que esta operação é utilizada tanto na fase de compressão, onde é precedida pela etapa de estimação de movimento (ME), quanto na fase de descompressão do vídeo. 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 na ME, localizar os blocos de melhor casamento na memória de quadros anteriormente codificados (quadros passados e/ou futuros) e montar o quadro predito. Na codificação, este quadro será subtraído do quadro atual, para gerar o quadro de resíduos que passará pela transformada.

Este trabalho apresenta a implementação em C de algoritmos para a compensação de movimento visando atender a alguns dos requisitos do padrão H.264. A partir da implementação em software, foram desenvolvidas arquiteturas para implementar o compensador de movimento em hardware, seguindo os algoritmos desenvolvidos. Foram implementados, primeiramente, algoritmos em linguagem C que tratam de tamanhos de blocos fixos. Estas primeiras versões manipulam tamanhos de bloco de 4x4, 8x8 ou 16x16 pixels. Após foi realizada a implementação para tamanhos de blocos variáveis sendo, possível a remontagem com os três tamanhos de blocos citados para o mesmo quadro do vídeo.

As arquiteturas desenvolvidas trabalham com tamanhos de blocos fixos de 4x4 ou 16x16 e também para tamanho de blocos variáveis utilizando os três tamanhos citados anteriormente. Estas arquiteturas processam vídeos de resolução QCIF (176X144 pixels). As arquiteturas foram descritas em VHDL e mapeadas para o dispositivo EP2S130F1020C4 da família Stratix II da Altera. Os resultados de

síntese obtidos indicam que as arquiteturas desenvolvidas seriam capazes de manipular vídeos HDTV em tempo real. Estas arquiteturas estão aptas a processar entre 455 e 607 quadros por segundo em uma resolução HDTV.

Palavras-chave: Compressão de vídeo, Compensação de movimento, Padrão H.264, VHDL, FPGA.

Abstract

SILVA, André Marcelo C. da **Desenvolvimento de um Compensador de Movimento para o Padrão H.264 de Compressão de Vídeo.** 2006. 63f. Monografia – Curso de Bacharelado em Ciência da Computação, Universidade Federal de Pelotas, Pelotas.

Digital videos are demanding an ever increasing amount of memory what renders difficult their transference, storage and manipulation. In order to alleviate this problem, video compression standards, as MPEG-1, MPEG-2, H.263, MPEG-4 and H.264 had been developed. Among these H.264 is the most recent video compression standard and its final draft was approved in October 2003.

An important operation used in video compression standards is motion compensation (MC). This operation is used in both coding and decoding, in the first case being preceded by the motion estimation operation (ME). operation. ME searches for the block with the best similarity in the memory of previously coded frames (past and/or future frames) and generates a motion vector for this block. The motion compensation is reconstructs the predicted frame using the motion vector. In coding, this reconstructed frame will be subtracted from the present frame to generate the frame residues, which is submitted to the transforms.

This work presents a C implementation of algorithms for motion compensation that have some characteristics of the H.264 pattern. Considering these software implementations, some architectures were designed to implement the motion compensation in hardware. The first versions of the software implementation considered just blocks with fix sizes of 4x4, 8x8 or 16x16 pixels. The last software implementation was able to handle blocks with variable size, where it is possible use the three block sizes in the reconstruction of the same video frame.

The designed architectures considered fix block sizes of 4x4 or 16x16 pixels and considered also the variable block sizes (4x4, 8x8 and 16x16 pixels), using a QCIF (176x144pixels) video resolution. The proposed architectures were described in VHDL and mapped into the EP2S130F1020C4 FPGA device from Altera Stratix II family. The syntheses results indicated that the designed architectures were able to process HDTV videos in real time. These architectures were able to process between 455 and 607 frames per second in HDTV resolution.

Key-Words: Video Compression, Motion Compensation, H.264 Standard, VHDL, FPGA

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ALUT	
AVC	*Advanced Video Coding*
CD	*Compact* Disk
CIF	*Common Intermediate Format*
CODEC	Codificador/Decodificador
DPCM	*Differential Pulse Code Modulation*
DVD	*Digital Versatile Disk*
FPGA	*Field Programmable Gate Array*
HD	*Hard Disk*
HDTV	*High Definition Digital Television*
HSI	*Hue, Saturation, Intensity*
JVT	*Joint Video Team*
MC	*Motion Compensation*
ME	*Motion Estimation*
MPEG	*Moving Pictures Experts Group*
QCIF	*Quarter Common Intermediate Format*
RGB	*Red, Green, Blue*
SBTVD	Sistema Brasileiro de Televisão Digital

SDTV	*Standard Definition Television*
VCEG	*Video Coding Experts Group*
VHDL	VHSIC *Hardware Description Language*
VHSIC	*Very High-Speed Integrated Circuits*
VGA	*Video Graphics Adapter*
VM	*Vetor de Movimento*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*

Sumário

1. Introdução

Com o desenvolvimento crescente de aplicações que manipulam imagens e vídeos, como câmeras digitais, TV digital, celulares, palmtops, entre outros, é cada vez mais relevante a capacidade dos sistemas de manipularem eficientemente estas imagens e vídeos.

Considerando aplicações que manipulam vídeos, o problema é muito desafiador, pois os arquivos dos vídeos estão cada vez maiores, devido à exigência na qualidade de imagens a serem transmitidas ou pelo fato dos vídeos possuirem uma longa duração. Isso faz com que tais vídeos demandem quantidades cada vez maiores de memória, dificultando sua transferência via rede e até mesmo, o seu armazenamento em disco. Além disso, a complexidade para processar estes vídeos em tempo real também é crescente. Considerando que a transmissão de vídeos digitais é um grande gargalo para o bom desempenho dos sistemas de transmissão e recepção de dados, então a compressão destes vídeos é essencial para viabilizar este tipo de aplicações.

A compressão de vídeos também é muito importante do ponto de vista de armazenamento, pois contribui para otimizar o uso dos recursos geralmente utilizados para tal finalidade, tais como DVDs, CDs, HDs etc. A transmissão em *broadcast* de vídeos digitais também é uma outra área onde a compressão é muito importante, pois viabiliza a transmissão de elevadas taxas de informação a uma taxa de bits reduzida. Este é o caso da TV Digital, que seria inviável se não usasse algoritmos eficientes para a compressão de vídeo (RICHARDSON, 2002).

Em função de tais problemas, foram desenvolvidos padrões de compressão de vídeo, tais como MPEG-1 (ISO/IEC, 1993), MPEG-2 (ITU-T, 1994), H.263 (ITU-T, 2000) e padrões mais novos, como MPEG-4 e H.264 (RICHARDSON, 2003), este

último tendo seu texto final aprovado em outubro de 2003 (JOINT VIDEO TEAM, 2003).

A compressão de vídeo é essencial para o sucesso das aplicações que manipulam vídeos digitais, pois um vídeo não comprimido utiliza uma quantidade de bits muito elevada para representar cada pixel da imagem. Isso implica em custos muito elevados em termos de armazenamento e transmissão destas informações e estes custos acabam por inviabilizar o desenvolvimento de produtos para esta área, caso a compressão não seja utilizada.

Uma importante operação utilizada nos padrões de compressão de vídeo é a compensação de movimento (CHU; ANASTASSIOU; CHANG, 1997), a qual é utilizada tanto na fase de compressão quanto na fase de descompressão do vídeo.

Neste sentido, é muito importante ter uma implementação para este bloco que corresponda às exigências de desempenho, utilização de recursos e de freqüência de operação, pois são fatores críticos no projeto deste bloco e conseqüentemente, para o projeto final do compressor do H.264 (RICHARDSON, 2003).

Este trabalho tem como objetivo estudar e implementar um compensador de movimento a ser utilizado na compressão, envolvendo a implementação em software, a proposição da arquitetura sua descrição em linguagem VHDL (IEEE, 1993), e sua validação..

O capítulo dois desta monografia apresenta conceitos básicos de compressão de vídeos digitais que serão de fundamental importância para a compreensão dos demais capítulos deste trabalho. No capítulo três são apresentados alguns conceitos do padrão H.264 e da compensação de movimento neste padrão. O capítulo quatro apresenta os algoritmos utilizados na implementação em software do compensador de movimento e os resultados desta implementação. O capítulo cinco apresenta a arquitetura do compensador de movimento, detalhando os blocos que o compõem, além de apresentar os resultados destas arquiteturas quando mapeadas para FPGAs. Para finalizar, o capítulo seis apresenta as conclusões e propostas de trabalhos futuros.

2. Conceitos Básicos da Compressão de Vídeo

Este capítulo apresentará, resumidamente, alguns conceitos básicos sobre compressão de vídeos digitais e também sobre a sua forma de representação, que facilitará o entendimento dos demais capítulos deste trabalho.

2.1 Conceitos Básicos do Vídeo Digital

Um vídeo digital é formado por uma seqüência de quadros (*frames*), que, por sua vez, são formados por pontos (*pixels*). Cada quadro do vídeo é dividido em blocos. O padrão H.264 é o primeiro que prevê a utilização de blocos com tamanhos variáveis (4x4, 4x8, 8x4, 8x8, 8x16, 16x8 e 16x16 pixels).

Em um vídeo digital existe muita redundância na informação decorrente da similaridade existente entre pontos vizinhos e também entre quadros vizinhos. Pontos vizinhos, são pontos que pertencem ao mesmo quadro e que são espacialmente vizinhos. Quadros vizinhos são quadros pertencentes ao mesmo vídeo e que são temporalmente vizinhos.

A compressão de vídeos é uma técnica que utiliza algoritmos (codecs) para diminuir o tamanho de uma imagem através da redução destas informações redundantes que formam a imagem.

2.2 Espaço de Cores e Sub-Amostragem 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

sensíveis à intensidade luminosa, enquanto os cones são sensíveis às cores primárias (GONZALEZ, 2003). Em função desta estrutura do sistema visual humano, todas as cores são vistas como combinações variáveis das três cores primárias: vermelho (R – *red*), verde (G – *green*) e azul (B – *blue*). O sistema visual humano é capaz de discernir milhares de cores distintas a partir de combinações de intensidades distintas das cores primárias. Por outro lado, o sistema visual humano não consegue distinguir mais do que duas dúzias de tons de cinza que, na verdade, indicam a intensidade luminosa da imagem (GONZALEZ; WOODS, 2003).

Existem muitas formas de 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, tendo em vista que é este o espaço de cores utilizado nos monitores coloridos. O RGB representa, em três matrizes distintas, as três cores primárias captadas pelo sistema visual humano: vermelho, verde e azul. Daí advém 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) (MIANO, 1999).

Os componentes R, G e B possuem um elevado grau de correlação, tornando difícil o processamento de cada uma das informações de cor de forma independente. Por isso, a compressão de vídeos 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 compressores de imagens estáticas e vídeos.

O sistema visual humano possui cerca de 240 milhões de bastonetes e 13 milhões de cones (GONZALEZ, 2003). Deste modo, o sistema visual humano é muito mais sensível a informações de luminância do que as informações de crominância. Desta forma, é conveniente que os padrões de compressão de imagens estáticas e vídeos explorarem esta característica psicovisual humana para

aumentar a eficiência de codificação, o que se dá através da redução da taxa de amostragem dos componentes de crominância em relação aos componentes de luminância (RICHARDSON, 2002). Esta operação é chamada de sub-amostragem de cores e é realizada sobre o espaço de cores YCbCr nos padrões de compressão de vídeos atuais.

Existem várias formas de relacionar os componentes de crominância com o componente de luminância para realizar a sub-amostragem. 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 sub-amostragem 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, a qual deveria ser 4:1:1 (RICHARDSON, 2003).

A sub-amostragem de cor aumenta significativamente a eficiência da compressão, uma vez que parte da informação da imagem é simplesmente descartada, sem causar impacto visual perceptível ao olho humano. 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 sub-amostragem.

O padrão H.264, considera que os dados do vídeo de entrada estão no espaço de cores YCbCr. Sub-amostragens de cor nos formatos 4:2:0, 4:2:2 e 4:4:4 são permitidos, mas o formato mais usado é o 4:2:0.

3. A Compensação de Movimento no Padrão H.264

Este capítulo traz um breve histórico e alguns conceitos sobre o padrão H.264 de compressão de vídeo. Também são abordados os detalhes sobre o bloco de compensação de movimento, o que ajudará a contextualizar melhor a relevância desta monografia.

3.1 Histórico do padrão H.264

O padrão de compressão de vídeo H.264, 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 foi desenvolvido pelo JVT (ITU-T, 2000) que foi formado a partir de uma união entre os especialistas do VCEG da ITU-T (ITU-T, 2003) e do MPEG da ISO/IEC (ISO/IEC, 1993). A primeira versão do H.264 foi aprovada em 2003.

O padrão H.264 foi desenvolvido por um período de aproximadamente quatro anos. As raízes deste padrão estão no projeto H.26L da ITU-T, que foi iniciado pelo VCEG (ITU-T, 2005), que construiu a chamada para propostas no início de 1998 e que criou o primeiro rascunho deste novo padrão 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 surgir, alguns outros padrões já haviam sido criados e já estavam consolidados, servindo de base para o desenvolvimento do H.264. O primeiro padrão relevante para a construção do H.264 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 por 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). Este padrão se tornou um padrão popular e é muito usado 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, 2005) finalizou o desenvolvimento do seu mais recente padrão, conhecido como MPEG-4 Parte 2 (ISO/IEC, 1999). Então, 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 rascunho 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 afirmaram as escolhas de desenvolvimento realizadas pelo VCEG no H.26L:

- 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.

Então, 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, 2005), 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 ITU-T 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 rascunho final (ITU-T, 2003) aprovado em outubro de 2003 (SULLIVAN, 2005).

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

3.2 Conceitos básicos do padrão H.264

O padrão H.264 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 o padrão H.264 em relação aos demais padrões disponíveis na atualidade. Este aumento de complexidade impede, pelo menos na tecnologia atual, a utilização de codecs H.264 implementados em software quando as resoluções são elevadas e/ou quando deseja-se tempo real, com 30 quadros por segundo, por exemplo. A intratabilidade do problema via software somado pelo 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.

Existem muitas aplicações potenciais para codecs H.264, 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 muitas destas soluções não estão reportadas em detalhes na literatura. Do ponto de vista da academia, existem muitas equipes espalhadas pelo mundo trabalhando com o H.264, buscando soluções de software e/ou hardware para atacar o problema da complexidade elevada do padrão. Vários trabalhos têm sido publicados nos últimos dois anos, mas a área encontra-se ainda repleta de problemas sem solução e, conseqüentemente, muitas contribuições

inovadoras poderão ser descobertas e implementadas. A fig. 3.1 mostra o diagrama de blocos do codificador do padrão H.264.

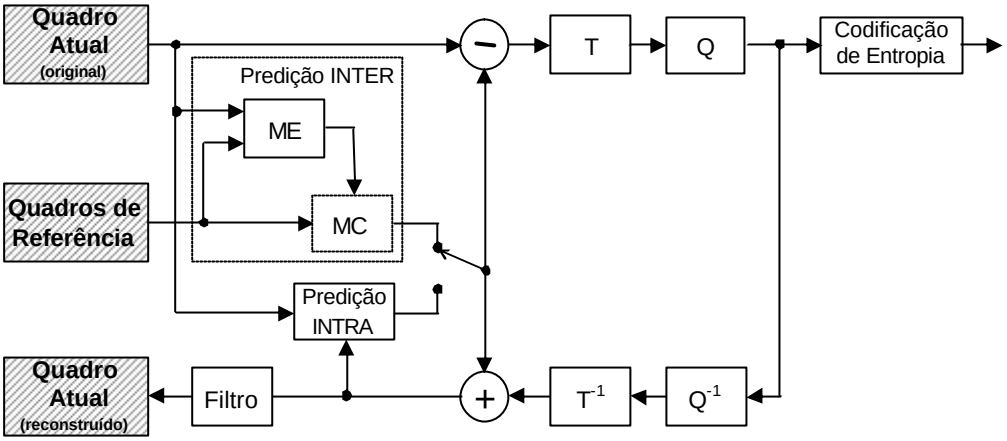

Figura 3.1 - Diagrama em blocos de um codificador H.264.

3.3 A estimação de movimento

A compensação de movimento (MC) está intimamente ligada à estimação de movimento (ME). É na estimação de movimento que são gerados os vetores de movimento (VM) que serão utilizados na compensação. Por isso, a MC deve adaptar-se às definições da estimação de movimento.

Quando analisamos um trecho de um vídeo, podemos perceber que as imagens vizinhas são muito similares. As diferenças, quando existem, são geradas em sua maioria, por movimentos da câmera ou de objetos pertencentes à cena. Esta característica dos vídeos produz uma grande redundância temporal, ou seja: se comparados dois quadros vizinhos, pode-se perceber uma enorme quantidade de informações repetidas, causadas por regiões dentro do quadro que não são alteradas de um quadro para outro.

O objetivo principal da estimação de movimento é reduzir esta redundância temporal entre imagens vizinhas. A técnica consiste em identificar estas redundâncias e mapear as diferenças através de vetores denominados de vetores de movimento. A partir de um ou mais quadros, denominados de quadros de referência, o quadro atual, que é o próximo quadro após o quadro de referência, é estimado usando vetores de movimento obtidos a partir do(s) quadro(s) de referência. Para cada bloco do quadro atual será gerado um vetor de movimento que

corresponderá a posição onde este bloco obteve a melhor relação de similaridade no quadro de referência (PORTO, 2006).

Existem vários algoritmos para realizar esta busca de melhor similiaridade. Contudo, neste trabalho foram utilizados VMs gerados pelo algoritmo FULL SEARCH (busca completa). O algoritmo de busca completa procura a melhor relação entre as áreas de pesquisa (*best matching*) comparando o bloco que está sendo pesquisado com todas as posições possíveis dentro da área de pesquisa. O bloco é deslocado de um pixel dentro da área de pesquisa, começando do canto superior esquerdo, até que todas as posições tenham sido comparadas e a menor diferença tenha sido registrada. Ao final, será gerado um vetor de movimento referente ao deslocamento do bloco para a região de melhor casamento (LIN, 2005).

3.4 A compensação de movimento

A compensação de movimento é o processo de compensação para o deslocamento de objetos movidos de um quadro para outro.

O Compensador de Movimento (MC) no codificador H.264 é uma etapa precedida pela etapa de Estimação de Movimento (ME). É função da compensação de movimento, a partir do vetor de movimento gerado na ME, localizar 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. 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.

É nas etapas de estimação e compensação de movimento (KUHN, 1999) (bloco ME e MC na fig. 3.1) onde se encontra a maior parte das inovações e dos ganhos obtidos pelo padrão H.264 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.

A principal inovação do H.264, do 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 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 Fig. 3.2 (WIEGAND, 2003; RICHARDSON, 2003), e então, é necessário que o compensador de movimento esteja apto a atender estas características do estimador de movimento.

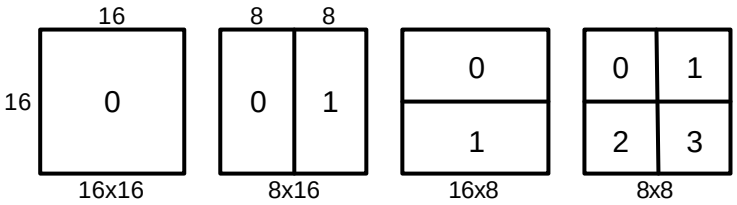

Figura 3.2: Divisão do macrobloco em partições de macroblocos

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 chamadas sub-macroblocos, que podem ter o tamanho 8x8, 4x8, 8x4 ou 4x4, como está apresentado na fig. 3.3.

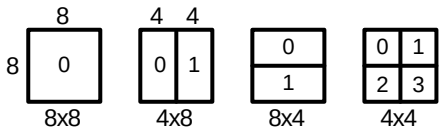

Figura 3.3: Divisão de uma partição de macrobloco em partições de sub-macroblocos.

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 16x16 de luminância corresponde a uma partição de 8x8 de crominância.

Um vetor de movimento é necessário para cada partição ou sub-macrobloco. A escolha de um tamanho de partição grande (16x16, 16x8, 8x16) implica na utilização de um número menor de bits para identificar o vetor de movimento escolhido e 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 de bits maior para representar o vetor de movimento e para identificar a partição escolhida. Como pode ser percebido, 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.

4. Compensador de Movimento em Software

A implementação do compensador de movimento em software foi realizada na linguagem de programação “C” (KERNIGHAN; RITCHIE, 1999) sendo que foram implementadas quatro alternativas para o MC. Em três destas alternativas foram utilizados blocos de tamanhos fixos de 4x4, 8x8 e 16x16 pixels, por serem estas três alternativas os extremos e o valor médio de tamanhos de blocos utilizados pelo padrão. Com isto, foi possível obter-se uma visão abrangente de como se comporta o resíduo resultante da reconstrução dos quadros para os diferentes valores de tamanho de bloco.

A quarta alternativa foi desenvolvida para tamanhos de blocos variáveis, o que significa que os três tamanhos de blocos utilizados na implementação para tamanhos fixos foram usados para a reconstrução de um mesmo quadro do bloco. Todas as implementações utilizaram o espaço de cores YCbCr com sub-amostragem de cores 4:2:0. Os vetores de movimento foram gerados pela estimação de movimento através do algoritmo *Full Search* (PORTO, 2006).

4.1 Implementação para blocos de tamanhos fixos

A primeira alternativa foi uma implementação utilizando o tamanho dos blocos de 16x16 pixels. Depois foram feitas as implementações 8x8 e 4x4 pixels, respectivamente.

Na literatura não são encontradas informações aprofundadas sobre a implementação do compensador de movimento e os algoritmos que são utilizados. Geralmente, as publicações existentes fazem somente referência às características que o compensador deve atender para satisfazer plenamente o padrão de

compressão de imagem que está sendo mencionado, não detalhando nenhum algoritmo específico. A única referência algorítmica é o texto do padrão H.264, onde existe toda a documentação.

O algoritmo utilizado para desenvolver estas alternativas de implementação não tomou como base qualquer outro algoritmo que esteja sendo utilizado ou estudado, sendo criado um novo algoritmo a partir das idéias básicas do funcionamento do compensador de movimento do padrão de compressão de vídeos H.264, observando suas principais características.

4.1.1 Implementação para tamanho de bloco de 16x16 pixels

No caso do compensador de movimento para blocos de tamanho de 16x16 pixels, o algoritmo utiliza-se de três arquivos binários para realizar suas funções. O primeiro arquivo a ser utilizado é o que possui os vetores de movimento, chamado de “FS_VCOMP_30”. Neste arquivo encontram-se os vetores de movimento para todos os quadros do vídeo digital. Estes vetores foram gerados pela estimação de movimento dos seus respectivos blocos 16x16 dos quadros que serão remontados. O vídeo utilizado para testes possui uma resolução de 720x480 pixels. Então, neste caso, a leitura de 1350 pares de valores de vetores de movimento é necessária para cada quadro, em um total de 351 mil acessos a este arquivo para a reconstrução total do vídeo, uma vez que este vídeo possui 260 quadros.

O segundo arquivo a ser utilizado é o que possui os quadros de referência e é chamado de “src21_ref__525_480i@30.yuv” (FEDORA, 2005). Este vídeo pode ser encontrado na internet e é utilizado para realizar testes sobre implementações que manipulam vídeos. Este arquivo contém os quadros do vídeo e é onde o algoritmo busca os valores correspondentes a cada pixel da imagem.

Por fim, o último arquivo utilizado pelo algoritmo é o arquivo “remontado_COMP_30_16x16_FS_V.yuv”, onde vão ser escritas as informações lidas do arquivo do quadro de referência para gerar o quadro remontado.

O algoritmo possui, primeiramente, um laço “for” que é responsável por contar o número de quadros do vídeo que se deseja remontar. Nesta implementação foi remontado o vídeo completo, ou seja, 260 quadros. Após isto, é chamada a função “carrega referencia” que é responsável por carregar nas matrizes específicas

de cada elemento da sub-amostragem de cores (YCbCr) os valores contidos no arquivo de referência.

A seguir, dois laços “for” fazem o controle do número de blocos que serão remontados em cada quadro do vídeo. Como se está usando uma resolução de 720x480 pixels e um tamanho de bloco de 16x16 pixels, então serão 45 blocos por linha e 30 blocos por coluna, em cada quadro. Na seqüência, é lido um par de valores do arquivo de VMs e estes valores são divididos por dois para serem utilizados na matriz dos elementos Cb e Cr (que são sub-amostrados). A seguir, é feita a passagem dos valores já deslocados, caso seja necessário, contidos nas matrizes carregadas com os valores de referência, para uma outra matriz temporária, que estará com os valores corretos para a remontagem do bloco.

Finalmente, é feita a passagem dos valores das matrizes temporárias de YCbCr para um arquivo do tipo “yuv”, que estará pronto para ser visualizado após a remontagem de todos os quadros do vídeo. A seguir é mostrado um pseudo-código deste algoritmo para facilitar o seu compreendimento.

```
repita(quadro = 0 até quadro = 99);
    carrega valores do arquivo de referencia;

    repita(alt = 0 até alt = 464);
      repita(larg = 0 até larg = 704);

        le arquivo de VMs;
        le arquivo de VMs;

        calcula pos. de leitura dos valores de Chroma no arq. de referencia;
        calcula pos. de leitura dos valores de Luma no arq. de referencia;

        repita(enquanto posição coluna no macrobloco de Luma = OK);
          repita(enquanto posição linha no macrobloco de Luma = OK);
            matriz temp de Luma <- valor de Luma do arq. de referencia;

        repita(enquanto posição coluna no macrobloco de Cb = OK);
          repita(enquanto posição linha no macrobloco de Cb = OK);
```

matriz temp de Cb <- valor de Cb do arq. referencia;

repita(enquanto posição coluna no macrobloco de Cr = OK);
repita(enquanto posição linha no macrobloco de Cr = OK);
matriz temp de Cr <- valor de Cr do arq. de referencia;

escreve valores da matriz temp de Luma no arq. remontado;
escreve valores da matriz temp de Cb no arq. remontado;
escreve valores da matriz temp de Cr no arq. remontado;

A diferença residual, após a codificação intraquadro ou interquadro, é obtida através de uma subtração dos valores dos macroblocos do quadro atual e dos valores gerados por estas codificações (RICHARDSON, 2003). Esta diferença é chamada de resíduo. O resíduo, então, é enviado para os blocos responsáveis por reduzir a redundância psicovisual (SHI, 99).

Neste trabalho, foi extraído o quadro de número 151 do vídeo original, que é o vídeo que está sendo codificado, e também o quadro 151 do mesmo vídeo após ter sido feita a compensação de movimento. Foi, então, realizada a subtração do quadro original pelo remontado e obteve-se o quadro do resíduo destes dois quadros. As figs. 4.1, 4.2 e 4.3 mostram o quadro original, o quadro remontado e o resíduo resultante da subtração para a matriz de luminância (Y), respectivamente.

Figura 4.1 – Quadro original.

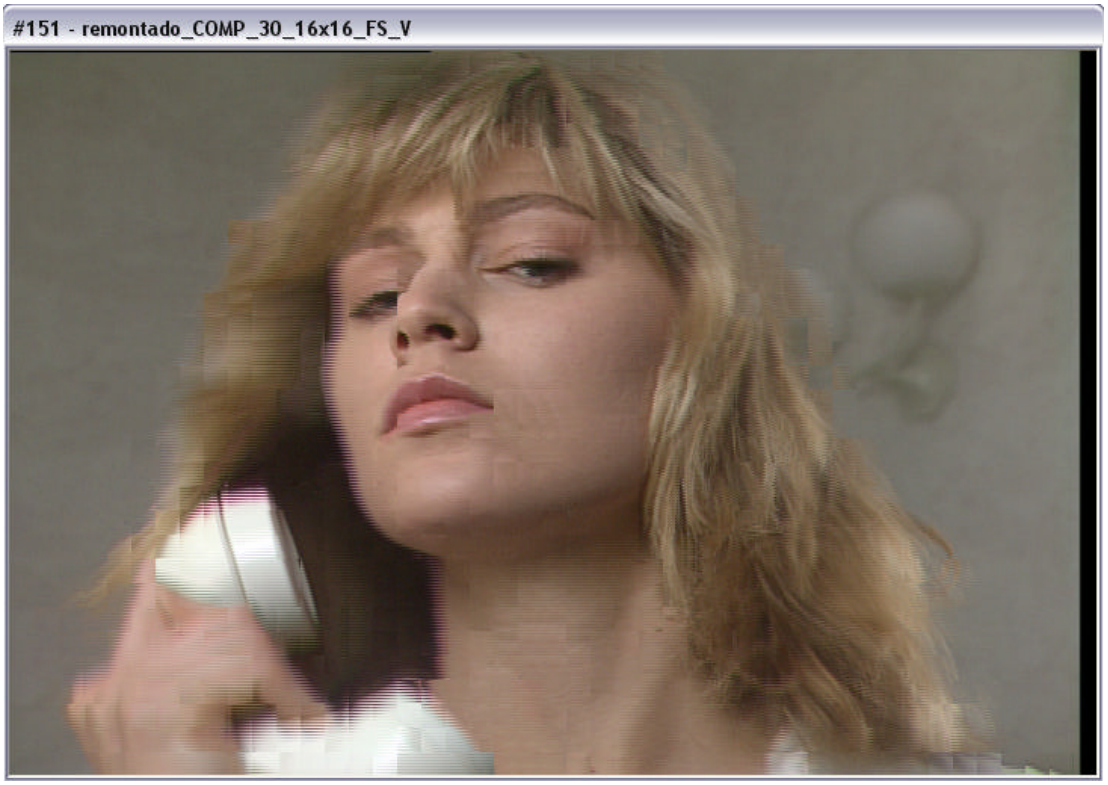

Figura 4.2 – Quadro remontado com blocos de 16x16 pixels.

Figura 4.3 – Quadro de luminância residual para remontagem com blocos de 16x16.

Nas figs. 4.1, 4.2 e 4.3, teve-se o cuidado de selecionar um quadro da seqüência que apresentava uma quantidade maior de movimento que os demais quadros do vídeo para poder mostrar nitidamente o efeito de bloco no quadro remontado e, também, mostrar o resíduo decorrente da subtração. Na maior parte dos quadros deste vídeo o resíduo é muito próximo de zero.

4.1.2 Implementação para tamanho de bloco de 8x8 pixels

A estrutura e a lógica utilizada na versão anterior foram preservadas nesta implementação. Porém, houve algumas mudanças significativas no algoritmo. Esta versão utilizou o mesmo arquivo de VMs que a versão de 16x16 pixels, porém, foi realizada uma adaptação neste arquivo para que ele contivesse o número de VMs necessários a esta implementação. Além dos dois laços “for” que já eram utilizados na versão anterior, para controlar os blocos de 16x16, foram acrescentados mais dois laços “for” para fazer o controle dos blocos 8x8 dentro de cada macrobloco 16x16, garantindo, assim a remontagem correta de cada bloco. Novamente, o quadro 151 (fig. 4.1) foi extraído do vídeo remontado. Também foi gerado um quadro

de resíduos de luminância para esta implementação. Estes quadros estão apresentados nas figs. 4.4 e 4.5, respectivamente.

Figura 4.4 – Quadro remontado com blocos de 8x8 pixels.

Figura 4.5 – Quadro de luminância residual para remontagem com blocos de 8x8.

4.1.3 Implementação para tamanho de bloco de 4x4 pixels

Nesta versão o algoritmo também é similar ao da versão 16x16, sobre o qual foram realizadas as adaptações necessárias para tratar este tamanho de bloco. Estas mudanças estão nos laços que controlam a cópia da matriz com os valores de referência para a matriz temporária. Nas figs. 4.6 e 4.7 estão mostrados os quadros 151 do vídeo remontado e do vídeo de resíduo gerado. O quadro original é o mesmo da fig. 4.1. Contudo, o efeito de bloco nesta figura fica pouco visível devido ao tamanho do bloco utilizado nesta versão ser muito pequeno.

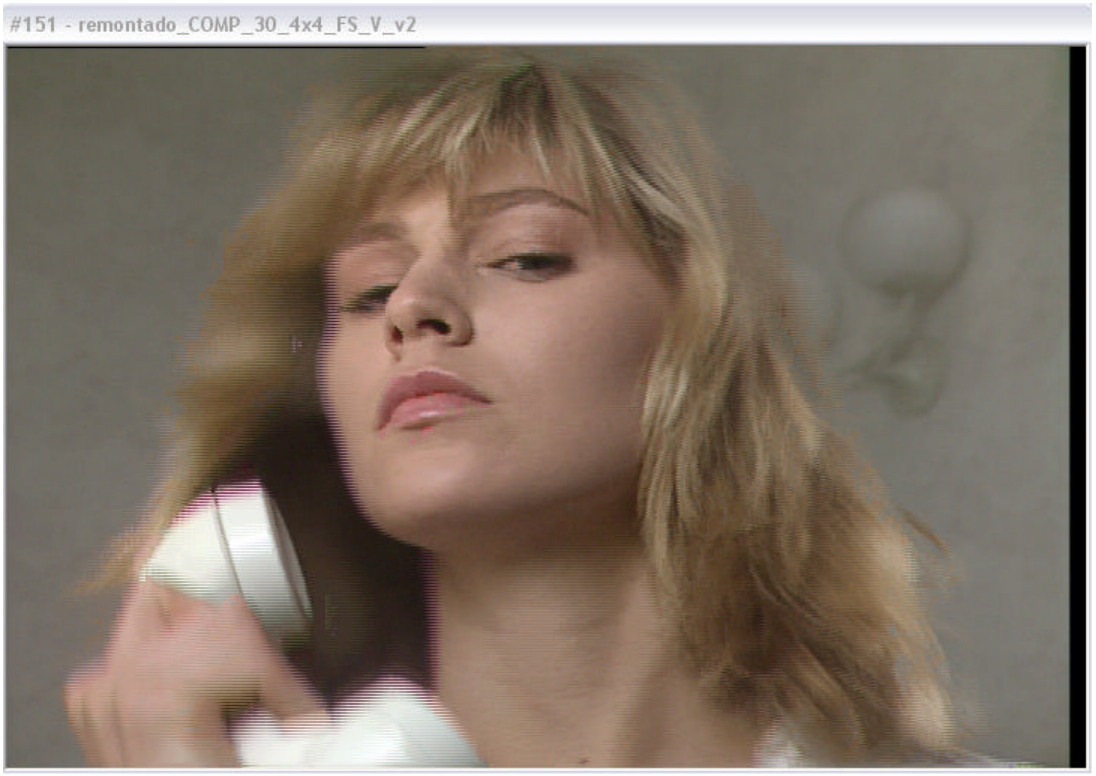

Figura 4.6 – Quadro remontado com blocos de 4x4 pixels.

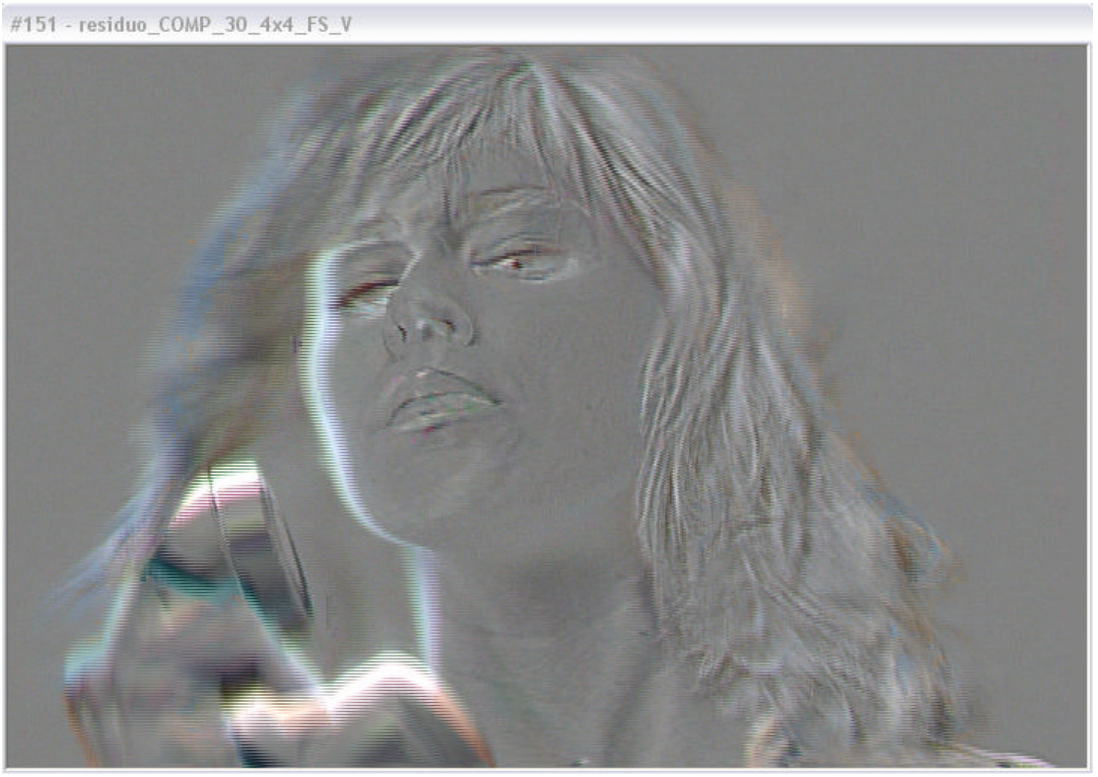

Figura 4.7 – Quadro de luminância residual para remontagem com blocos de 4x4.

4.2 Diferença entre os resíduos gerados

Para se ter uma idéia da diferença entre o resíduo gerado em implementações com diferentes tamanhos de blocos, foi realizada a subtração entre os vídeos de resíduos, gerando assim um novo vídeo. Como era esperado, a versão com blocos de 16x16 pixels gerou um resíduo maior que as demais versões. Portanto, os vídeos foram construídos subtraindo do resíduo desta implementação (16x16 pixels) do resíduo gerado nas outras duas implementações (8x8 pixels e 4x4 pixels).

A tab. 4.1 mostra a diferença absoluta, ou seja, a soma total entre a subtração de cada pixel dos quadros dos vídeos de diferenças de resíduos

Tabela 4.1 – Diferença absoluta entre os resíduos gerados.

	16x16 para 8x8	**16x16 para 4x4**
Diferença	139.525.008	210.595.376

Nas figs. 4.8 e 4.9 são mostrados os resultados do mesmo quadro 151, mas agora para o vídeo de diferenças de resíduos, com a subtração de cada quadro do vídeo de resíduos gerado em cada versão.

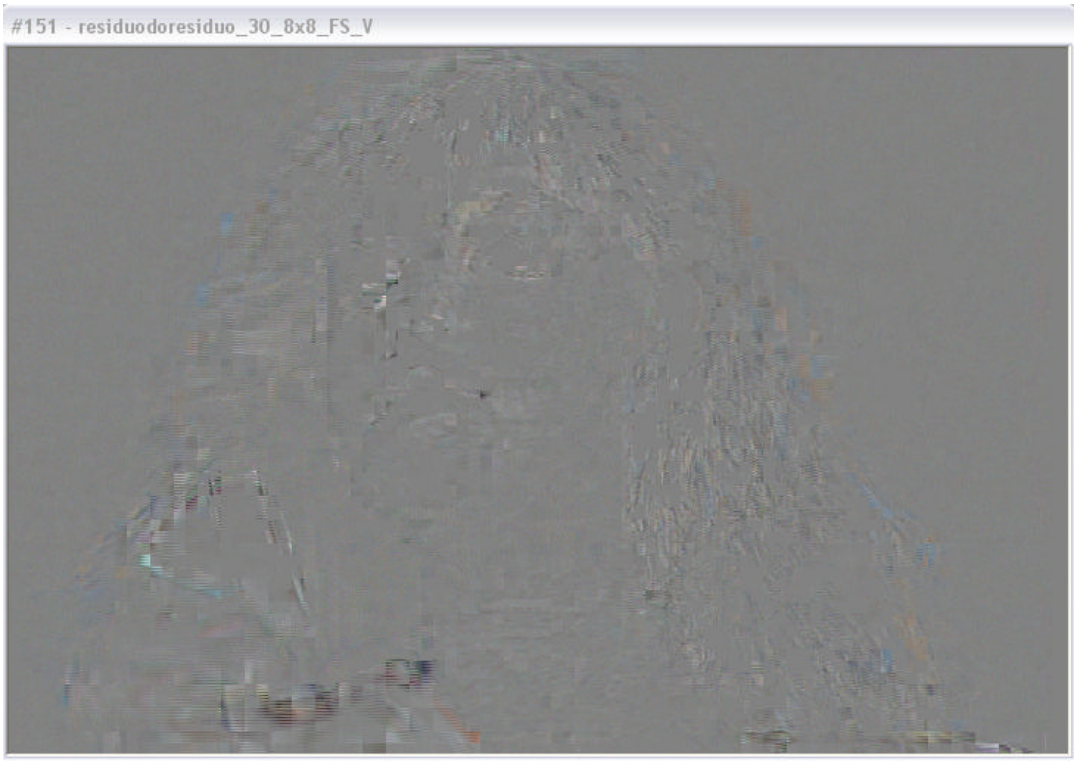

Figura 4.8 – Diferença entre resíduos (16x16 para 8x8).

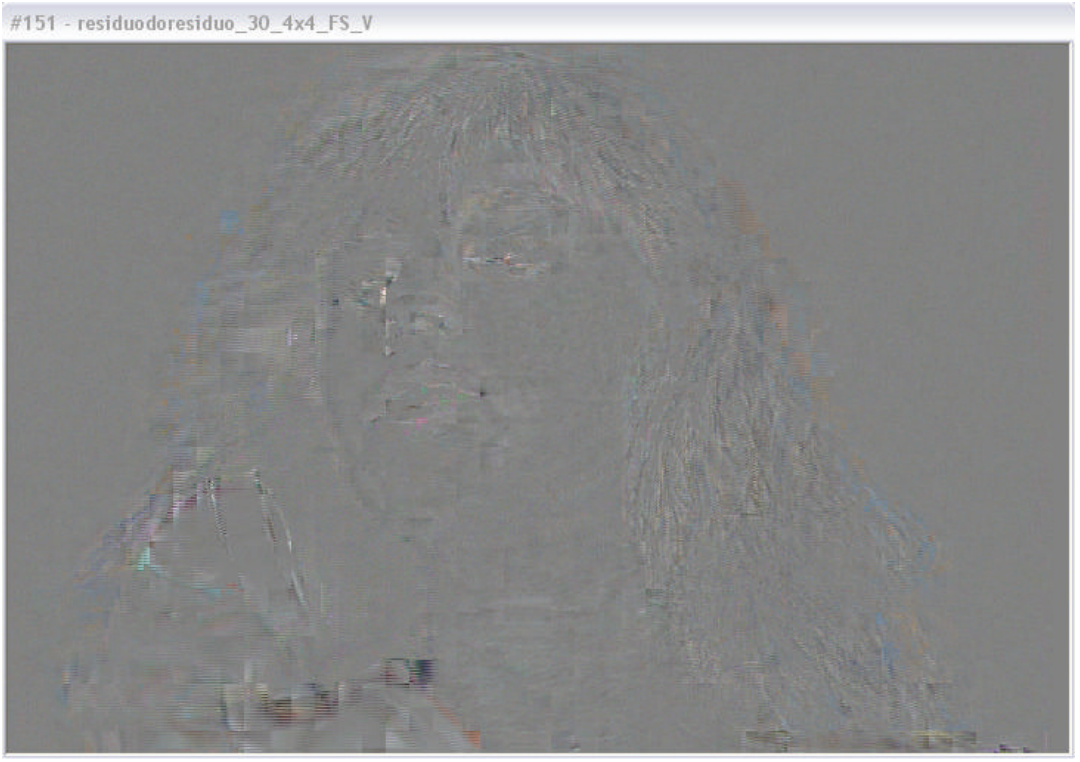

Figura 4.9 – Diferença entre resíduos (16x16 para 4x4).

Nestas figuras, é possível identificar visualmente que a fig. 4.9 possui uma diferença maior em relação à implementação 16x16, do que a fig. 4.8, indicando que o tamanho de bloco 4x4gera o menor resíduo dentre todos os tamanhos de blocos, conforme já era esperado.

4.3 Implementação para tamanho de blocos variáveis

Esta implementação começa, também, carregando os três arquivos necessários para a sua execução. Esta versão também possui um laço “for” para controlar o número de quadros do vídeo a ser reconstruído. Porém, a partir deste ponto, ela pouco se assemelha às demais implementações, pois necessitou de uma complexidade relativamente maior para ser implementada.

O primeiro passo foi dividir cada macrobloco de 16x16 pixels em 16 blocos de 4x4 pixels. Então, estes blocos foram numerados para viabilizar o controle dos laços utilizados no algoritmo e que serão detalhados nos próximos parágrafos. A fig. 4.4 mostra esta divisão de um bloco em macroblocos com a numeração utilizada.

Uma variável chamada “localizador” indica qual destes blocos está sendo remontado pelo algoritmo, que antes de realizar a remontagem, leu do arquivo de VMs a informação de qual tipo de bloco será tratado. Esta informação de tipo de bloco indica se o bloco é do tipo 1 (16x16 pixels), do tipo 2 (8x8 pixels) ou do tipo 3 (4x4 pixels). Se o bloco for tipo 1, então é realizada a remontagem do bloco através do algoritmo utilizado na versão 16x16 e a variável “localizador” vai para a posição 15, indicando que o bloco foi totalmente remontado.

	4	4	4	4
4	0	1	2	3
4	4	5	6	7
4	8	9	10	11
4	12	13	14	15

Figura 4.10 – Macrobloco de 16x16 pixels dividido em 16 blocos de 4x4 pixels.

Se o bloco for do tipo 2, então é realizada a remontagem deste bloco com o algoritmo utilizado na versão 8x8, e a variável “localizador” assumirá uma entre quatro possibilidades. Se a variável “localizador” possuir o valor zero, então será remontado o primeiro bloco 8x8 dos quatro blocos 8x8 existentes no macrobloco 16x16. A variável “localizador” passará a possuir o valor 2 após a remontagem. Nesse momento, as variáveis “alt” e “larg” são atualizadas com os respectivos valores de altura e largura do próximo bloco 8x8. Os valores de altura e largura indicam a posição de cada bloco dentro do macrobloco. Por exemplo, o bloco zero na fig. 4.10 possui valores de altura e largura iniciais iguais a zero e valores finais de 3. Já o bloco 1 possui a altura inicial zero e a largura inicial 3. Porém, após a remontagem de um bloco 8x8, os valores são atualizados para a posição do macrobloco 2 (altura zero e largura 12), para que na próxima remontagem seja feita a partir deste ponto. Se a variável “localizador” possuir o valor 2, ela passará a possuir o valor 8. Se a variável contiver o valor 8, passará a conter o valor 10. Finalmente, se a variável possuir o valor 10, passará a possuir o valor 15, indicando que o bloco de 16x16 foi totalmente remontado.

Após um bloco do tipo 2 (8x8 pixels) ser processado, o próximo bloco será um outro 8x8 ou quatro blocos 4x4, exceto quando o bloco 8x8 for o último de um macrobloco 16x16. Neste caso, o próximo bloco poderá ser de 16x16. Neste caso, a variável “localizador” irá receber o valor 15.

Se o bloco for do tipo 3 (4x4 pixels) então é possível prever que outros três blocos deste tamanho serão processados para remontar um macrobloco de 8x8, porém, em cada remontagem é feita a atualização dos valores de altura e largura e também da variável “localizador”. O pseudo-código abaixo exemplifica esta implementação.

```
repita(quadro = 0 até quadro = 99);
    carrega valores do arquivo de referencia;

    repita(alt = 0 até alt = 464);
        repita(larg = 0 até larg = 704);
            variavel localizador <- 0;

            enquanto localizador != 15 faça
```

```
                le o tamanho do bloco do arq. de VM;

se tipo = 1 (bloco 16x16);

        le arq. de VM;
        le arq. de VM;

        calcula pos. de leitura dos valores de Chroma no arq. de referencia;
        calcula pos. de leitura dos valores de Luma no arq. de referencia;

        repita(enquanto posição coluna no macrobloco de Luma = OK);
            repita(enquanto posição linha no macrobloco de Luma = OK);
                matriz temp de Luma <- valor de Luma do arq. de referencia;

        repita(enquanto posição coluna no macrobloco de Cb = OK);
            repita(enquanto posição linha no macrobloco de Cb = OK);
                matriz temp de Cb <- valor de Cb do arq. referencia;

      repita(enquanto posição coluna no macrobloco de Cr = OK);
          repita(enquanto posição linha no macrobloco de Cr = OK);
                matriz temp de Cr <- valor de Cr do arq. de referencia;

variavel localizador <- 15;

se tipo = 2 (bloco 8x8);

        le arq. de VM;
        le arq. de VM;

        calcula pos. de leitura dos valores de Chroma no arq. de referencia;
        calcula pos. de leitura dos valores de Luma no arq. de referencia;

        repita(enquanto posição coluna no macrobloco de Luma = OK);
            repita(enquanto posição linha no macrobloco de Luma = OK);
```

```
                matriz temp de Luma <- valor de Luma do arq. de referencia;

        repita(enquanto posição coluna no macrobloco de Cb = OK);
                repita(enquanto posição linha no macrobloco de Cb = OK);
                        matriz temp de Cb <- valor de Cb do arq. referencia;

        repita(enquanto posição coluna no macrobloco de Cr = OK);
                repita(enquanto posição linha no macrobloco de Cr = OK);
                        matriz temp de Cr <- valor de Cr do arq. de referencia;

se localizador = 0;
        então localizador <- 2;

se localizador = 2;
        então localizador <- 8;

se localizador = 8;
        então localizador <- 10;

se localizador = 10;
        então localizador <- 15;

se tipo = 3 (bloco 4x4);
        repita(quadro 1 até quadro 4);

        le arq. de VM;
        le arq. de VM;

        calcula pos. de leitura dos valores de Chroma no arq. de referencia;
        calcula pos. de leitura dos valores de Luma no arq. de referencia;

        repita(enquanto posição coluna no macrobloco de Luma = OK);
                repita(enquanto posição linha no macrobloco de Luma = OK);
```

```
                    matriz temp de Luma <- valor de Luma do arq. de
referencia;

        repita(enquanto posição coluna no macrobloco de Cb = OK);
            repita(enquanto posição linha no macrobloco de Cb = OK);
                matriz temp de Cb <- valor de Cb do arq. referencia;

        repita(enquanto posição coluna no macrobloco de Cr = OK);
            repita(enquanto posição linha no macrobloco de Cr = OK);
                matriz temp de Cr <- valor de Cr do arq. de referencia;

    se localizador = 0
        então localizador <- 2;

    se localizador = 2
        então localizador <- 8;

    se localizador = 8
        então localizador <- 10;

    se localizador = 10
        então localizador <- 15;

    localizador <- 0;

    escreve valores da matriz temp de Luma no arq. remontado;
    escreve valores da matriz temp de Cb no arq. remontado;
    escreve valores da matriz temp de Cr no arq. remontado;
```

4.4 Resultados da Implementação

Esta seção tem o objetivo de apresentar os resultados obtidos com estas implementações realizadas em software para os quatro casos propostos neste trabalho. A tab. 4.2 mostra o número de quadros por segundo que cada versão

implementada é capaz de processar, ressaltando que a resolução utilizada foi a de SDTV (720 x 480 pixels), com estes valores de taxa de processamento, como já era esperado, nenhuma das versões do MC em software atingiu a taxa de processamento exigida para aplicações que processam vídeos desta resolução em tempo real (30 quadros por segundo).

Tabela 4.2 – Taxa de processamento das implementações, em quadros/s.

		8x8	4x4	VARIÁVEL
Quadros/Segundo (SDTV)	6,03	5,88	5,57	5,44

Processador Intel Celeron 2.8 GHz, 256MB de memória principal

A partir dos valores da tab. 4.2 foi realizada uma estimativa da taxa de processamento para outras resoluções de vídeo. A tab. 4.3 ilustra o resultado destas estimativas.

Estes resultados mostram que apenas a implementação do MC em software para a resolução QCIF atinge a taxa de processamento de 30 quadros por segundo. Neste caso, a implementação alcança uma taxa de processamento que varia de 74,22 a 82,27 quadros por segundo entre o pior e o melhor caso, respectivamente. Outro dado importante é a taxa de processamento para HDTV, onde o MC atinge a taxa de apenas um quadro por segundo no melhor caso.

Tabela 4.3 – Estimativa da taxa de processamento das implementações para outras resoluções, em quadros/s.

	16x16	8x8	4x4	VARIÁVEL
QCIF (176x144)	82,27	80,21	75,99	74,22
CIF (352x268)	22,07	21,52	20,39	19,91
VGA (640x480)	6,78	6,62	6,28	6,12
HDTV (1920x1080)	1,00	0,98	0,92	0,90

Nas duas tabelas apresentadas nota-se que não existe uma grande diferença no tempo de processamento quando se altera o tamanho do bloco. A maior diferença está na resolução QCIF. Porém, esta resolução atinge facilmente altas taxas de processamento.

5. Hardware para o Compensador de Movimento

Foram desenvolvidas três arquiteturas principais para a implementação da compensação de movimento. A primeira delas considerou apenas blocos de tamanho fixo com 16x16 pixels. A segunda arquitetura também considerou blocos de tamanho fixo, porém com 4x4 pixels. Finalmente, a terceira implementação considerou blocos de tamanho variável, de 4x4, 8x8 ou 16x16 pixels. Não foi desenvolvida uma versão com blocos de tamanho fixo de 8x8 pixels. A versão inicial das três soluções, por uma questão de simplicidade, considerou apenas informações de luminância, ou seja, os vídeos de entrada eram representados em tons de cinza, sem cores. Com a primeira versão funcional e validada, uma segunda versão, considerando as informações de cores, foi desenvolvida.

As três implementações foram baseadas no diagrama de blocos apresentado na fig. 5.1. A fig. 5.1 mostra o uso de quatro blocos de memória, um para armazenar as informações de luminância do frame de referência, outro para armazenar as informações de crominância do frame de referência e outros dois blocos para as informações de luminância e crominância do frame remontado. Além disso, está presente na fig. 5.1 o bloco que representa o núcleo do compensador de movimento. A memória utilizada foi mapeada para a memória interna do dispositivo FPGA, o que representa uma grande vantagem do ponto de vista do tempo de acesso.

Para permitir a implementação em hardware, foram utilizados frames tipo QCIF, pois, para frames que apresentam maiores resoluções, não haveria memória suficiente no dispositivo.

As arquiteturas foram descritas em VHDL e sintetizadas para o dispositivo EP2S130F1020C4 da família Stratix II da Altera. Este dispositivo foi escolhido,

fundamentalmente, por possuir memória interna suficiente para implementar as arquiteturas.

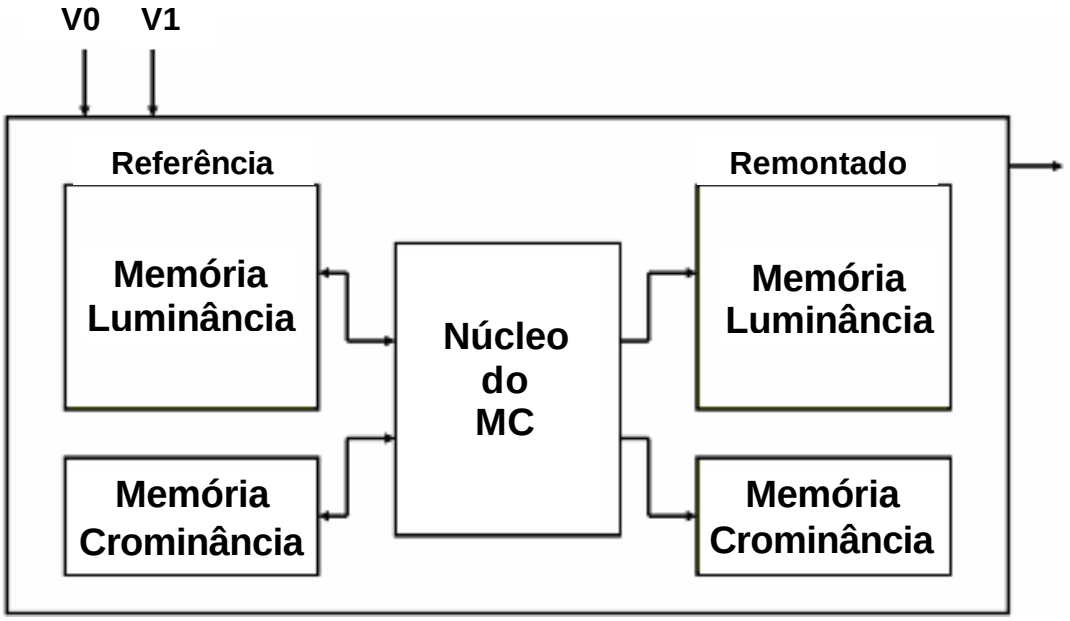

Figura 5.1 – Diagrama de Blocos da Arquitetura do MC.

5.1 – Arquitetura para blocos de tamanho fixo de 16x16 pixels

Nesta arquitetura, a memória implementada foi organizada em palavras de 128 bits, sendo que as memórias de luminância possuem 1584 palavras, enquanto as memórias de crominância possuem 792 palavras. Cada palavra representa uma linha de macrobloco como mostra a fig. 5.2. Desta forma, obtém-se uma otimização no número de leituras necessárias para remontagem do quadro. A leitura das informações de crominância é feita somente nas linhas pares do bloco. Para armazenar os frames QCIF (176x144 pixels), cada memória precisa de 304.128 bits (2376 palavras de 128 bits).

Com esta organização, onde cada palavra representa uma linha de cada macrobloco, quando houver um vetor que indique a necessidade de um deslocamento na horizontal, é possível que seja necessária a solicitação de uma leitura extra para que a palavra seja montada corretamente, pois parte da informação necessária para a remontagem do macrobloco estará nesta nova leitura realizada. O valor contido no vetor de movimento é multiplicado por oito para saber o número exato de bits que deverão ser considerados na releitura realizada. Havendo um deslocamento na vertical, a leitura do bloco de referência começa a ser realizada

a partir da linha indicada com o valor contido no vetor de movimento (VM). Por exemplo: se o VM contiver o número 3, a linha inicial de contagem é 3.

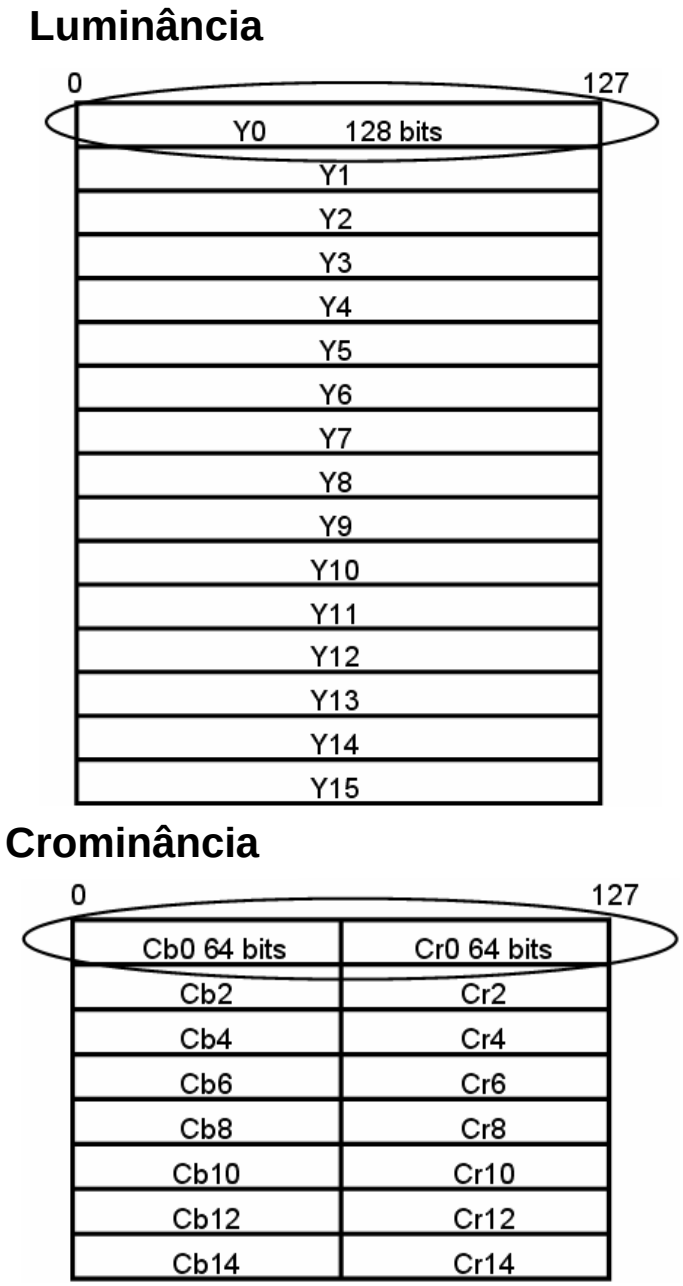

Figura 5.2 – Organização da memória para o MC para blocos 16x16 pixels.

A interface do núcleo do compensador de movimento para a arquitetura que considera blocos de tamanhos fixos com 16x16 pixels está apresentada na fig. 5.3. Este núcleo possui as entradas “Vetor 0” e ”Vetor 1”, que possuem os valores correspondentes ao vetor de movimento do macrobloco e servem para calcular o deslocamento do macrobloco, caso seja necessário. A entrada “Dado_Ref” contém o

endereço da palavra de memória do quadro do vídeo que está sendo utilizado como quadro de referência.

A saída “End_Ref” contém o endereço da palavra de memória do quadro de referência em que vai ser realizada a leitura dos dados. Isto ocorre quando a saída “Lê_Ref” está com valor 1, habilitando assim a leitura dos dados.

A saída “End_Rem” contém o endereço de escrita da palavra de memória no quadro remontado. Esta escrita dos dados é realizada quando a saída “Esc_Rem” possui o valor 1 e, então, os dados são passados pela saída “Dado_Rem”. A saída “OK” indica que o macrobloco foi remontado sem erro.

Figura 5.3 – Núcleo do MC.

O núcleo do MC para blocos fixos de 16x16 pixels foi desenvolvido baseado na máquina de estados representada na fig. 5.4. No estado inicial, estado 0, a máquina obtém os valores dos vetores de movimento gerados pela estimação de movimento para cada macrobloco de tamanho fixo 16x16. Ainda neste estado, o compensador faz o cálculo do primeiro endereço de leitura de memória necessário e encaminha para o próximo estado, que é o estado 1. No estado 1 o compensador verificará se será necessária uma nova leitura, ou seja, se o vetor de movimento indicou um deslocamento na horizontal. Caso positivo, a leitura do novo endereço é solicitada e a execução passa para o estado 2. Caso contrário, a execução passa para o estado 3, que é o estado onde será realizada a escrita.

O estado 2 é o estado no qual o compensador de movimento precisa ficar em modo de espera, enquanto é realizada a segunda leitura na memória. Posteriormente, a execução é encaminhada para o estado 3. No estado 3, será realizado o cálculo do endereço de escrita e a montagem da palavra (caso necessário) e, então, é solicitada a escrita na memória do frame remontado e a

execução volta para o estado inicial, até que se conclua a montagem das 16 palavras que compõe um macrobloco.

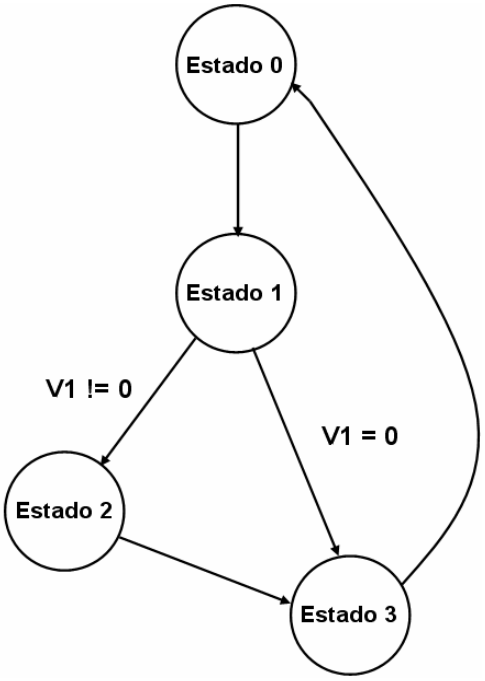

Figura 5.4 – Máquina de estados para o núcleo do MC considerando blocos fixos de 16x16 pixels

A tab. 5.1 apresenta os resultados de síntese para esta arquitetura, considerando apenas as informações de luminância. Na tab. 5.2 estão os resultados para a implementação considerando também as informações de crominância. Estas tabelas apresentam os resultados dos blocos de memória, do núcleo do compensador de movimento e do bloco completo.

Tabela 5.1 - Resultados de síntese sem as informações de crominância para blocos fixos de 16x16 pixels.

	Memória	Núcleo do MC	Bloco Completo
Freqüência (MHz)	309	196	93
ALUTs	197	3223	3636
Registradores	4	304	312
Bits de	202.752	-	405,504

Tabela 5.2 - Resultados de síntese com as informações de crominância para blocos fixos de 16x16 pixels.

	Memória Luminância	Memória Crominância	Núcleo do MC	Bloco Completo
Freqüência (MHz)	309	309	189	87
ALUTs	-	-	1519	1542
Registradores	-	-	581	579
Bits de Memória	202.752	101.376	-	608.256

Como a versão arquitetural que considera as informações de cor precisa utilizar as informações de crominância, então esta versão da arquitetura utiliza 1,5 vezes mais bits de memória que a versão mais simplificada, que considera somente informações de luminância. O bloco de memória, com a implementação sem as informações de crominância, utilizou 405.504 bits de memória, enquanto a versão que considerou as informações de crominância utilizou 608.256 bits de memória para os dois blocos de memória necessários.

O núcleo do compensador alcançou uma freqüência de operação de 196 MHz na implementação sem informações de cores e uma freqüência de 189 MHz na implementação que considera informações de crominância. Com estas freqüências, o núcleo MC na implementação sem as informações de crominância é capaz de processar até 504 quadros HDTV por segundo no melhor caso, e até 486 quadros HDTV por segundo na versão utilizando as informações de crominância. O bloco completo também seguiu esta tendência, que era esperada, qual seja: a implementação com cores alcançou uma freqüência menor que a implementação sem cores. Para tratar informações coloridas, foi necessário modificar a estrutura de geração de endereços, o que fez com que esta estrutura ficasse mais complexa. Além disso, na versão colorida, é necessário o acesso à memória que contém os elementos de crominância, aumentando o tempo necessário para concluir a operação. O MC completo para blocos fixos de 16x16 pixels atingiu uma freqüência de 93 MHz sem as informações de crominância, o que resulta em um processamento de 237 quadros HDTV por segundo e utilizou 3% dos recursos de memória do dispositivo. Já para a implementação com as informações de cores, o bloco completo conseguiu uma freqüência de 87 MHz, que significa um taxa de

processamento de 223 quadros HDTV por segundo, utilizando 9% dos recursos de memória do dispositivo. Todos os valores de taxa de processamento citados são para vídeos de alta resolução (HDTV), com 1920x1080 pixels.

5.2 Arquitetura para blocos de tamanho fixo de 4x4 pixels

Para a implementação com blocos de tamanhos fixos de 4x4 pixels, foi identificado que a melhor organização de memória para os elementos de luminância era com palavra de 128 bits, onde cada palavra representa um bloco 4x4. Para os elementos de crominância, a melhor organização de memória era com palavras de 64 bits, onde cada leitura realizada contém os elementos de um bloco de Cb e outro bloco de Cr, como mostra a fig. 5.5. Quando não há deslocamento no bloco ou quando o deslocamento é múltiplo de 4 pixels, então será necessária apenas uma leitura na memória que contém o frame de referência. Nos casos em que há deslocamento somente na horizontal ou somente na vertical e onde o deslocamento não é múltiplo de 4 pixels, então são necessárias duas leituras na memória. Nos casos em que há deslocamento na horizontal e na vertical que não são múltiplos de 4 pixels, então são necessárias quatro leituras na memória.

A fig. 5.6 mostra a máquina de estados desenvolvida para implementar o núcleo do compensador de movimento, que controla as leituras necessárias na memória do frame de referência e a escrita na memória do frame remontado. No estado 0, o compensador recebe os vetores de movimento e calcula o endereço da primeira leitura.

No estado 1, é realizado um teste para verificar se serão necessárias novas leituras. Caso positivo, o endereço da próxima leitura é calculado e a execução é encaminhada para o estado 2. Caso contrário, a execução já é levada para o estado 5, que é o estado de escrita. No estado 2, será verificado se há necessidade de mais leituras. Se houver, o próximo endereço é calculado e o próximo estado será o estado 3. Se não houver necessidade de nova leitura, o próximo passo será a escrita (estado 5). No estado 3, já há a certeza de que uma nova leitura deverá ser solicitada, então é procedida a solicitação. O estado 4 é o estado que é usado apenas para aguardar a conclusão da leitura da memória solicitada no estado anterior. Então, a execução é encaminhada para o estado 5. No estado 5 é calculado o endereço do bloco 4x4 a ser remontado e a montagem da palavra

conforme as leituras realizadas e, então, é solicitada a escrita na memória do frame remontado.

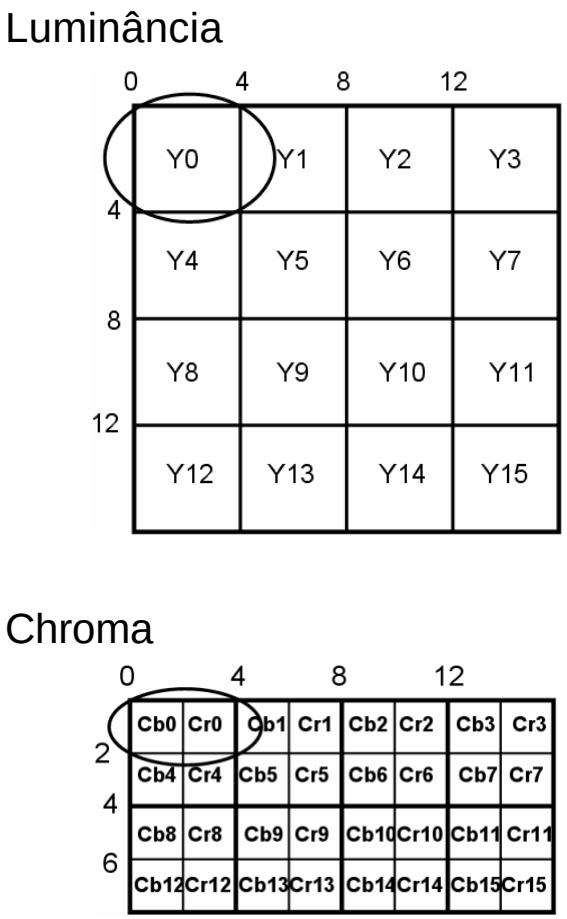

Figura 5.5 – Organização da memória para o MC para blocos 4x4.

A tab. 5.3 apresenta os resultados de síntese para a memória, o núcleo do compensador e bloco completo da compensação de movimento para blocos de tamanho fixo de 4x4 pixel, desconsiderando as informações de Cb e Cr. A tab. 5.4 mostra os resultados considerando as informações de crominância. A quantidade de bits de memória necessários para armazenar os frames de referência e remontado foram os mesmos que foram utilizados na implementação anterior, tendo em vista que estão sendo considerados vídeos com o mesmo formato (QCIF). O núcleo do compensador atingiu a freqüência de operação de 125 MHz e o bloco completo atingiu a freqüência de 90 MHz na primeira versão, sem considerar informações de cores. Com estes resultados de freqüência obtidos foram feitas estimativas da taxa de processamento de vídeos com alta resolução (HDTV) com 1920x1080 pixels.

O núcleo do compensador pode processar até 80 quadros por segundo e o bloco completo pode processar 58 quadros por segundo, utilizando 2% dos recursos de memória do dispositivo. Na segunda implementação, que considera as informações de cores, os valores das freqüências obtiveram quedas pouco significativas (caíram para 122MHz e 85MHz, respectivamente), que significa uma taxa de processamento de 78 quadros HDTV por segundo para o núcleo do MC e de 55 quadros HDTV por segundo para o bloco completo. Estes resultados indicam que tanto o núcleo do MC como o bloco completo de ambas versões da implementação podem ser utilizadas para processar vídeos de alta definição em tempo de real.

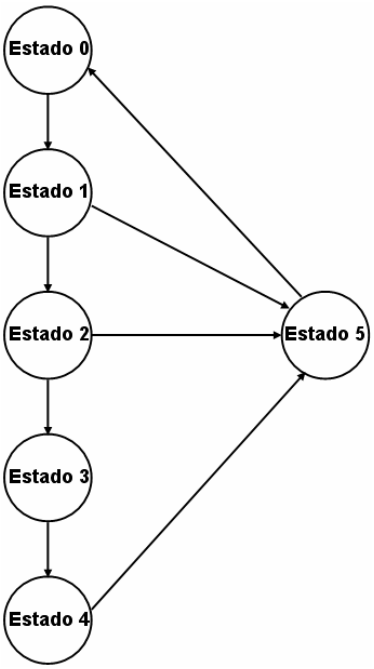

Figura 5.6: Máquina de estados para o núcleo do MC considerando blocos fixos de 4x4 pixels.

Comparando a solução que trata blocos fixos com 16x16 pixels com esta solução que trata blocos fixos de 4x4 pixels, a freqüência diminui em ambas as versões desta implementação, pois é necessário um aumento no número de acessos à memória de quadros e também um aumento no número de cálculos realizados pelo núcleo do compensador, quando o tamanho de bloco é reduzido para 4x4 pixels.

Porém, esta queda de freqüência de operação já era esperada, devido ao aumento no número de leituras e escritas na memória realizadas nesta implementação, decorrente da redução do tamanho do bloco.

Tabela 5.3 - Resultados de síntese sem as informações de crominância para blocos fixos de 4x4 pixels.

	Memória	Núcleo do MC	Bloco Completo
Freqüência (MHz)	350	120	90
ALUTs	197	1796	2.206
Registradores	4	453	465
Bits de Memória	202.752	-	405.504

Tabela 5.4 - Resultados de síntese com as informações de crominância para blocos fixos de 4x4 pixels.

	Memória Luminância	Memória Crominância	Núcleo do MC	Bloco Completo
Freqüência (MHz)	309	309	122	85
ALUTs	-	-	2378	2413
Registradores	-	-	624	624
Bits de Memória	202.752	101.376	-	608.256

5.3 Arquitetura para blocos de tamanhos variáveis

A versão arquitetural que manipula tamanhos de blocos variáveis considera que um mesmo quadro pode ser remontado com três tamanhos diferentes de blocos, 4x4, 8x8 e 16x16. Para a implementação desta arquitetura, foi usada uma organização de memória igual à arquitetura da fig. 5.5. O núcleo do compensador foi desenvolvido através de uma máquina de estado de formato similar ao apresentado na fig. 5.6. As diferenças residem na inserção de controles adicionais para tratar as possibilidades de posicionamentos de blocos de formato diferente dentro de um macrobloco e também, a inserção da identificação do tipo do próximo bloco que vai

ser remontado. A fig. 5.7 apresenta o diagrama em bloco do núcleo do compensador de movimento para utilização de blocos variáveis.

As entradas deste compensador de movimento são as mesmas da versão de 16x16 pixels, que é mostrada na fig. 5.3. Apenas foi acrescentada a entrada “Tipo”, que contém a informação referente ao tamanho do bloco que irá ser remontado: se é tipo 1 (16x16 pixels), tipo 2 (8x8 pixels) ou tipo 3 (4x4 pixels). Nas saídas também ocorreu somente o acréscimo do sinal “Erro” que indica a impossibilidade de remontar o bloco.

Figura 5.7: Diagrama de blocos do núcleo do MC.

O funcionamento da máquina de estados para leituras e escritas é idêntico ao da fig. 5.6. Apenas foi necessária a inclusão de um controle no estado 0 para confirmar se o tipo de bloco indicado na entrada está em uma posição permitida. Por exemplo, quando há indicação de um bloco do tipo 16x16, este só poderá ser remontado se a posição atual for o início de um macrobloco, ou seja, não é possível ter um macrobloco 16x16 entrando em um macrobloco que possui dois blocos 4x4 já remontados. Caso seja detectado algum problema desse tipo, um sinal de erro é ativado.

Nas tabs. 5.5 e 5.6 estão os resultados de síntese, em termos de utilização de recursos e freqüência de operação. Esta versão da arquitetura também foi desenvolvida para os dois casos, com e sem informação de crominância.

Com relação às outras implementações realizadas neste trabalho, o núcleo do compensador de movimento teve uma redução na sua freqüência de operação. Já o bloco completo do MC manteve a freqüência praticamente inalterada. O uso de registradores aumentou com relação ao uso nas implementações anteriores, bem como o uso de ALUTs, devido ao aumento da complexidade do problema tratado.

Na versão que desconsidera as informações de crominância, as taxas de processamento para HDTV obtidas com estas freqüências são de 76 quadros por segundo para o núcleo do MC e de 57 quadros por segundo para o bloco completo.

Já para a versão que considera as informações de crominância, a taxa de processamento no núcleo do compensador de movimento foi igual à implementação sem as informações de crominância, porém sendo reduzida para 55 quadros por segundo no bloco completo, e utilizando 2% dos recursos de memória do dispositivo.

Esta versão também fica com a taxa de processamento acima do mínimo exigido para ser utilizada em HDTV de tempo real: 25 a 30 quadros por segundo,.

Tabela 5.5 - Resultados de síntese sem as informações de crominância para blocos de tamanhos variáveis.

	Memória	**Núcleo do MC**	**Bloco Completo**
Freqüência (MHz)	309	118	89
ALUTs	197	1810	2.219
Registradores	4	465	473
Bits de Memória	202.752	-	405.504

Tabela 5.6 - Resultados de síntese com as informações de crominância para blocos de tamanhos variáveis.

	Memória Luminância	**Memória Crominância**	**Núcleo do MC**	**Bloco Completo**
Freqüência (MHz)	309	309	118	86
ALUTs	-	-	2390	2427
Registradores	-	-	636	636
Bits de Memória	202.752	101.376	-	608.256

5.4 Resultados Comparativos

Esta seção mostra os resultados de processamento, em quadros por segundo, das implementações em hardware desenvolvidas para imagens coloridas, ou seja, considerando os elementos de crominância das imagens.

Foram considerados dois casos distintos. O melhor caso é quando o bloco do frame é montado utilizando o menor número possível de estados da máquina de estados, enquanto que o pior caso é quando é necessário passar por todos estados da máquina de estados para remontar o quadro. A tab. 5.7 mostra os resultados de taxa de processamento, considerando quadros SDTV (720x480pixels), das três implementações em hardware (considerando o melhor e o pior caso) e das implementações em software equivalentes.

Tabela 5.7 – Taxa de processamento de quadros SDTV das diversas implementações em software e em hardware.

	16x16	**4x4**	**VARIÁVEL**
Quadros/Segundo (SDTV) Hardware, Melhor Caso	1.344	328	664
Quadros/Segundo (SDTV) Hardware, Pior Caso	1.007	164	332
Quadros/Segundo (SDTV) Software	6,03	5,57	5,44

Os resultados contidos na tab. 5.7 mostram o quanto é mais rápida a implementação em hardware, na comparação com a implementação em software. No pior caso, as versões 16x16, 4x4 e variável das implementações em hardware são 167, 30 e 61 vezes mais rápidas que as versões equivalentes em software.

Considerando o melhor caso das versões em hardware, elas são 222, 59 e 122 vezes mais rápidas que as implementações equivalentes em software.

Estes resultados mostram que o ganho em processamento com a implementação em hardware é relevante e não pode ser desprezado, principalmente quando se pretende processar vídeos de alta resolução em tempo real.

As tabs. 5.8 e 5.9 apresentam os resultados das estimativas de taxa de processamento das implementações em hardware para algumas outras resoluções

de vídeos. A tab. 5.8 apresenta as estimativas, em quadros por segundo, considerando o melhor caso, enquanto que a tab. 5.9 apresenta as estimativas, em quadros por segundo, considerando o pior caso.

Tabela 5.8 – Estimativas de taxa de processamento, em quadros por segundo, das implementações em hardware, considerando o melhor caso.

	16x16	**4x4**	**VARIÁVEL**
QCIF (176x144)	18.518	4.484	9.090
CIF (352x268)	4.926	1.202	2.433
VGA (640x480)	1.510	369	746
HDTV (1920x1080)	224	55	110

Tabela 5.9 – Estimativas de taxa de processamento, em quadros por segundo, das implementações em hardware, considerando o pior caso.

	16x16	**4x4**	**VARIÁVEL**
QCIF (176x144)	13.888	2.237	4.524
CIF (352x268)	3.690	600	1.216
VGA (640x480)	1.133	184	373
HDTV (1920x1080)	168	28	55

Mesmo no pior caso, as implementações atingiram resultados muito satisfatórios, pois, além das aplicações para resoluções baixas com QCIF e CIF obterem altas taxas de processamento, as aplicações para resolução VGA obtiveram uma taxa mínima de 184 quadros por segundo utilizando blocos de 4x4 pixel. Para HDTV também se obteve uma taxa de processamento que torna viável sua utilização para aplicações de tempo real (25 a 30 quadros por segundo) para todas as versões da implementação.

Considerando o melhor caso, foram obtidas taxas de processamento ainda mais elevadas, chegando a 18.518 quadros por segundo para o formato QCIF com blocos de 16x16. Além disso, todas as taxas de processamento obtidas são muito superiores àquelas exigidas para tratar vídeos HDTV em tempo real.

Sendo assim, o compensador de movimento implementado em hardware pode ser utilizado para todos os formatos com aplicações em tempo real, pois todas as versões apresentaram taxas de processamento superiores ao mínimo exigido.

6. Conclusões

Este trabalho apresentou o estudo e desenvolvimento de algoritmos para o compensador de movimento do padrão H.264. Estes algoritmos foram implementados em linguagem C e os resultados destas implementações foram apresentados neste trabalho. A partir dos algoritmos implementados em C, foram desenvolvidas arquiteturas para o compensador de movimento. Estas arquiteturas foram descritas em VHDL e sintetizadas para FPGAs da Altera. A descrição das arquiteturas desenvolvidas e os resultados de síntese foram apresentados neste trabalho.

O padrão H.264 possui uma codificação muito complexa e, por isso, é importante que sejam desenvolvidas arquiteturas em hardware para os diversos blocos de um codificador e de um decodificador H.264. Este trabalho explorou algumas possibilidades de desenvolvimento para o compensador de movimento, partindo de implementações em software até implementações em hardware direcionadas para FPGAs. Em função da elevada complexidade do padrão, já era esperado que as implementações em software do compensador de movimento não atingissem taxas de processamento suficientes para tratar vídeos de alta resolução, como HDTV, em tempo real. Estas implementações em software são capazes de processar, no melhor caso, um quadro HDTV por segundo, o que está muito distante dos 25 a 30 quadros por segundo necessários para a sua utilização em codecs HDTV. Estas implementações em software atingem tempo real somente para quadros do tipo QCIF. Porém este formato possui uma resolução muito baixa.

Os algoritmos implementados serviram de base para o desenvolvimento das arquiteturas, fornecendo resultados satisfatórios para um entendimento das operações realizadas pelo compensador de movimento, além de servir como referência para trabalhos futuros que poderão ser desenvolvidos nesta área.

Considerando as arquiteturas que foram desenvolvidas para tratar imagens coloridas, todas atingiram taxas de processamento que permitem sua utilização em codecs HDTV para tempo real. A única implementação que ficou abaixo dos 30 quadros por segundo foi a versão com tamanhos fixos de 4x4 pixels (no pior caso), que atingiu uma taxa de processamento de 28 quadros por segundo. Ainda assim esta taxa é considerada tempo real, pois está na faixa de 25 a 30 quadros por segundo, que é utilizada nos diversos padrões existentes (GONZALES, 2003).

Em função do que foi exposto, é possível concluir que este trabalho obteve os resultados esperados. As implementações em software alcançaram os objetivos de desvendar o compensador de movimento do padrão H.264. Por outro lado, as implementações em hardware atingiram as principais metas, que eram de obter um compensador de movimento capaz de tratar múltiplos tamanhos de blocos e de atingir tempo real para resoluções elevadas.

Referências

ALTERA Corporation, “Altera: The Programmable Solutions Company”. Disponível em <http://www.altera.com>, Acesso em: mar 2006.

BROWN, Stephen; VRANESIC, Zvonko. **Fundamemtals of Digital Logic With VHDL Design**, 2. ed. New York: Mc Graw-Hill, 2005. 939p.

CHU, Chung-Tao; ANASTASSIOU, Dimitris; CHANG, Shih-Fu. **Hierarchical global motion estimation/compensation in low birate vídeo coding.** In: INT. SYMP. ON CIRCUITS AND SYSTEMS, Hong Kong, 1997. Proceedings...[S.I.]: IEEE Press, 1997, p. 1149-1152.

FEDORA Core Test Page. Disponível em: <http://ginfo05.dee.ime.eb.br/>, Acesso em dez 2005.

GONZALEZ, R.; WOODS, R. **Processamento de Imagens Digitais**. São Paulo: Edgard Blücher, 2003.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **1076-1993 IEEE Standard VHDL Language Reference Manual**, 1993.

ISO/IEC JTC1, “**Coding of moving pictures and associated audio for digital storage media up to about 1.5Mbit/s – Part 2: Video**”, ISO/IEC11172 (MPEG-1), Novembro de 1993.

ISO/IEC 14496-2 **- MPEG-4 Part 2 (01/1999): coding of audio visual objects – part 2: visual**. [S.l.], 1999.

ISO - International Organization for Standardization. Disponível em: <http://www.iso.org>. Acesso em: dez. 2005.

ITU-T. **Recommendation H.261 v1 (11/90): video codec for audiovisual services at px64 kbit/s**. [S.l.], 1990.

ITU-T e ISO/IEC JTC1, "**Generic coding of moving pictures and associated audio information – Part 2: Video**", ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG-2), Novembro de 1994.

ITU-T. "**Video coding for low bit rate communication**", ITU-T Rec. H.263 v1: Novembro de 1995, v2: Janeiro de 1998, v3: Novembro de 2000.

ITU-T.**"Recommendation H.264 (05/03): advanced video coding for generic audiovisual services"**. [S.l.], 2003.

ITU-T.**"Recommendation H.264 (03/05): advanced video coding for generic audiovisual services"**. [S.l.], 2005.

Joint Video Team (JVT) do ITU-T e ISO/IEC JTC1. **Advanced video coding for generic audiovisual services (ITU-T Rec. H.264 ou ISO/IEC 14496-10 AVC).** 2003, 303p.

KERNIGHAN, Brian W.; RITCHIE, Dennis M. **C: a Linguagem de Programação Padrão Ansi.** Rio de Janeiro: Campus, 1999.

KUHN, P. **Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation**. Boston: Kluwer Academic Publishers, 1999.

LIN, C.; LEOU, J. **An Adaptative Fast Full Search Motion Estimation Algorithm for H.264**. In: ISCAS 2005 - IEEE INTERNATIONAL SYMPOSIUM CIRCUITS AND SYSTEMS. **Proceedings…** Kobe: IEEE, 2005a, p. 1493-1496.

MIANO, J. **Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP**. New York: Assison-Wesley, 1999.

PORTO, Marcelo. **Investigação de Algoritmos e Desenvolvimento Arquitetural Para a Estimação de Movimento em Compressão de Vídeo Digital.** Pelotas, 2006. 71f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal de Pelotas.

RICHARDSON, I. **H.264 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.

SCHILDT, Herbert. **C Completo e Total**. 3. ed. São Paulo: Makron Books, 1996. 827p.

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.

WIEGAND, T.; et all. **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, Jul. 2003.

ANEXO A – Publicações Obtidas Durante o Curso de Graduação

Durante o curso de graduação foram desenvolvidos trabalhos de pesquisa junto ao GACI, Grupo de Arquiteturas e Circuitos Integrados, que foram publicados em eventos nacionais e internacionais. Alguns destes trabalhos não fazem nenhuma relação com esta monografia, porém todos eles foram de fundamental importância, pois formaram uma sólida base para o desenvolvimento deste trabalho e também para outros trabalhos futuros. Este anexo apresenta estas publicações.

Artigos Publicados em Eventos Internacionais:

1-Título: Exploração do Espaço de Projeto da Hadamard 4x4 Direta do Padrão de Compressão de Vídeo H.264/AVC.

Autores: SILVA, André Marcelo Coelho da; SILVA, Thaísa Leal da; PORTO, Marcelo Schiavon; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luís Almada; SILVA, Ivan Saraiva; BAMPI, Sergio; AGOSTINI, Luciano Volcan.

Evento: XII WORKSHOP IBERCHIP , San José, Costa Rica.

Ano: 2006.

2-Título: Impactos do Uso de Diferentes Arquiteturas de Somadores em FPGAs Altera.

Autores: SILVA, André Marcelo Coelho da; SILVA; PORTO, Marcelo Schiavon; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luís Almada; BAMPI, Sergio; AGOSTINI, Luciano Volcan.

Evento: XI WORKSHOP IBERCHIP , Salvador, Brasil.

Ano: 2005.

Artigos Publicados em Eventos Nacionais:

1-Título: *Design Space Exploration on the H.264 Forward 4x4 Hadamard Transform in H.264/AVC Video Compression Standard.*

Autores: SILVA, André Marcelo C. da; SILVA, Thaísa; PORTO, Marcelo S.; PORTO, Roger; GÜNTZEL, Jóse Luís Almada; SILVA, Ivan; BAMPI, Sérgio; AGOSTINI, Luciano Volcan.

Evento: XXI SIM – SIMPÓSIO SUL DE MICROELETRÔNICA – Porto Alegre – Brasil.

Ano: 2006.

2-Título: *Motion Compensation Architecture for Video Compression.*

Autores: REDIESS, Fabiane; SILVA, André Marcelo C. da; GÜNTZEL, José Luís; AGOSTINI, Luciano Volcan.

Evento: XXI SIM – SIMPÓSIO SUL DE MICROELETRÔNICA – Porto Alegre – Brasil.

Ano: 2006.

3-Título: *Hardware Para o Compensador de Movimento do Padrão H.264 de Compressão de Vídeo.*

Autores: SILVA, André Marcelo C. da; GÜNTZEL, José Luís; AGOSTINI, Luciano Volcan.

Evento: XIV CIC – CONGRESSO DE INICIAÇÃO CIENTÍFICA – UFPel – Pelotas - Brasil.

Ano: 2005.

Prêmio Jovem Pesquisador: Segundo Lugar na Área de Engenharias.

4- Título: *Impacts of Different Adder Architectures in Designs Directed to FPGAs.*

Autores: SILVA, André Marcelo C. da; PORTO, Marcelo S.; GÜNTZEL, José Luís Almada; AGOSTINI, Luciano Volcan

Evento: XX SIM – SIMPÓSIO SUL DE MICROELETRÔNICA – Santa Cruz do Sul – Brasil.

Ano: 2005.

5- Título: *Exploração do Espaço de Projeto de Hardware para as Transformadas Discretas do Padrão de Compressão de Vídeo H.264.*

Autores: ROSA, Leandro Zanetti Paiva da; PORTO, Roger Endrigo Carvalho; PORTO, Marcelo Schiavon; SILVA, Thaísa Leal da; SILVA, André Marcelo Coelho da; AGOSTINI, Luciano Volcan.

Evento: XIV CIC - CONGRESSO DE INICIAÇÃO CIENTÍFICA - UFPel, Pelotas - Brasil.

Ano: 2005.