UNIVERSIDADE FEDERAL DE PELOTAS
Programa de Pós-Graduação em Ciência da Computação

Dissertação

Otimizações Algorítmicas e Desenvolvimento Arquitetural para as DCTs do HEVC

Ricardo Garcia Jeske

Pelotas, 2013

RICARDO GARCIA JESKE

OTIMIZAÇÕES ALGORÍTMICAS E DESENVOLVIMENTO ARQUITETURAL PARA AS DCTs DO HEVC

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Luciano Volcan Agostini

Coorientador: Prof. Dr. Júlio Carlos Balzano de Matos

Pelotas, 2013

Dados Internacionais de Publicação (CIP)

J58o Jeske, Ricardo Garcia
Otimizações algorítmicas e desenvolvimento arquitetural para as DCTs do HEVC / Ricardo Garcia Jeske; Luciano Volcan Agostini, orientador; Júlio Carlos Balzano de Matos, coorientador. – Pelotas, 2013.
107 f.: il.

Dissertação (Mestrado em Computação), Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas. Pelotas, 2013.

1.Codificação de vídeo. 2.Otimização algorítmica. 3.HEVC. 4.Projeto em FPGA. 5.DCT 2-D de múltiplos tamanhos. I. Agostini, Luciano Volcan, orient. II. Matos, Júlio Carlos Balzano de, coorient. III. Título.

CDD: 004.22

Catalogação na Fonte: Leda Cristina Peres Lopes CRB:10/2064
Universidade Federal de Pelotas

Banca examinadora:

.....
Prof. Dr. Luciano Volcan Agostini (Orientador)

.....
Prof. Dr. Júlio Carlos Balzano de Matos (Coorientador)

.....
Prof. Dr. José Luís Almada Güntzel

.....
Prof. Dr. Marcelo Schiavon Porto

.....
Prof. Dr. Bruno Zatt

AGRADECIMENTOS

Com a conclusão desta dissertação, também finalizo uma etapa muito importante em minha vida. Este trabalho de Mestrado agregou não somente novos conhecimentos acadêmicos, mas também várias experiências e crescimento pessoal.

Agradeço primeiramente a Deus, por ter me iluminado e me guiado a este “destino”, através de uma série de encontros, desencontros e decisões em optar por este ou aquele caminho, que me conduziram até aqui.

À minha mãe, Ana Maria, não somente por todo o carinho e amor que sempre estiveram presentes, mas em especial pela demonstração de força em superar um problema de saúde nestes últimos anos. A tua luta e vitória me incentivaram a superar todos e quaisquer obstáculos encontrados pelo caminho, mostrando o quão pequenos eram cada um deles quando comparados à tua superação.

Ao meu pai, Nelson Jeske (*in memoriam*). Através de teus exemplos, aprendi muito e hoje carrego comigo muito do teu caráter. Obrigado pela educação e valores morais que me foram ensinados. Foi uma honra ter compartilhado desta tua jornada aqui na Terra como teu filho.

À minha esposa, amiga, confidente, amante..., Aline Neuschrank. Te agradeço por todo apoio e companheirismo no decorrer destes anos em que estamos juntos. E também por compreender a minha ausência em vários momentos, em virtude do desenvolvimento deste trabalho. Serás recompensada. ;-) Te amo!

As minhas irmãs, Marcia e Sandra, que sempre me incentivaram e me apoiaram para que eu conseguisse concluir mais esta etapa importante. Não posso deixar de citar aqui também o cunhado Daniel, que já havia protestado por não estar presente nos agradecimentos do TCC da Graduação! ☺

Ao meu orientador, Prof. Luciano Agostini. Em primeiro lugar, por ter me aceitado como seu orientando e assim me proporcionar esta experiência de vida extremamente gratificante que foi o Mestrado. Em segundo, também por todos os ensinamentos repassados no decorrer da pesquisa, bem como os esclarecimentos prestados de forma clara e objetiva sempre que solicitado. Foi um privilégio ter trabalhado ao teu lado! És uma pessoa “do bem”, acessível e extremamente competente. Sabes exercer tua liderança de forma não autoritária, o que é extremamente motivador. Valeu “meu patrão!” ☺

Ao Ruhan e ao Cláudio, bolsistas do PET, que me acompanharam e auxiliaram no decorrer de todo o desenvolvimento deste trabalho. Foi um prazer trabalhar ao lado de pessoas inteligentes e competentes como vocês. Como o Luciano já havia dito, formamos uma boa equipe!

Aos professores das cadeiras que cursei, em especial ao Luciano e ao Marilton pela didática de suas aulas, bem como aos demais professores do PPGC, em especial aqueles com quem tive mais contato e convívio durante a pesquisa: Rafael, Leomar, Júlio e Felipe.

Aos colegas da primeira turma de Mestrado do PPGC, em especial ao Vladimir, por toda a parceria e ajuda, e ao Henrique, pela troca de ideias durante a pesquisa.

A todos os bolsistas do GACI, os quais me proporcionaram um convívio extremamente agradável no laboratório, além de toda a ajuda fornecida pela troca de ideias e informações neste grande grupo de pesquisa, do qual me sinto honrado em fazer parte.

À secretária do PPGC, Marta, que me atendeu com simpatia, grande presteza e agilidade sempre que solicitada.

Ao Prof. Luís Cléber Carneiro Marques (IFSul), por ter me indicado e apresentado ao meu orientador. Eu já estava decidido a sair da cidade em busca de emprego na área de Automação, mas, através daquela tua ligação, deu-se início um novo rumo aos acontecimentos. Hoje continuo aqui em Pelotas, próximo aos meus familiares e amigos, concluindo o Mestrado e sendo teu colega como Prof. do IFSul. Ainda bem que atendi aquela ligação! ☺

Fico extremamente satisfeito por estar concluindo esta etapa e não levar comigo apenas o título de Mestre, mas também valorosos conhecimentos, já que fui inserido no mundo da pesquisa e mergulhei nesta fascinante área que é a codificação de vídeos. Além disso, tive a oportunidade de vivenciar o quão gratificante é trabalhar em um grupo com espírito de equipe, comecei a me aventurar na arte da escrita e revisão de artigos e ainda pude desenvolver as habilidades da oratória em apresentações de trabalhos. Sobretudo, levo comigo boas amizades que se formaram durante a pesquisa.

Finalmente, agradeço a todos que participaram, direta ou indiretamente, desta importe e gratificante etapa da minha vida. Muito obrigado!

RESUMO

JESKE, Ricardo G. **Otimizações Algorítmicas e Desenvolvimento Arquitetural para as DCTs do HEVC.** 2013. 107f. Dissertação - Mestrado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

A codificação de vídeos é uma área em constante evolução, já que são cada vez mais comuns os dispositivos capazes de processar vídeos digitais. A eficiência de um codificador de vídeo é dada pela relação entre taxa de bits e qualidade do vídeo comprimido e, para ampliar a eficiência nestes requisitos cruzados, os padrões de codificação de vídeo têm utilizado ferramentas cada vez mais complexas, gerando um elevado custo computacional. Este trabalho está focado nas ferramentas de codificação de vídeos do padrão HEVC, que ainda está em desenvolvimento. Mais especificamente, este trabalho está focado nas transformadas discretas do cosseno de tamanho variável definidas pelo padrão HEVC. São definidos quatro tamanhos de transformadas DCT 2-D no HEVC: 32x32, 16x16, 8x8 e 4x4, e estas transformadas, além de ser uma das novidades do padrão, também ampliam a sua eficiência e complexidade. Este trabalho apresenta contribuições em duas frentes: (a) no desenvolvimento de um algoritmo capaz de realizar otimizações nos algoritmos das DCTs (com a transformação de multiplicações em somas e deslocamentos e com o compartilhamento de subexpressões) e a implementação em software deste algoritmo para automatizar a geração das otimizações e (b) o desenvolvimento em hardware dos algoritmos otimizados, visando baixo custo e elevada taxa de processamento. Estas contribuições são relevantes especialmente quando dispositivos móveis, alimentados por bateria, são considerados. As otimizações propostas permitiram, no melhor caso (DCT 32x32) uma redução expressiva no número de operações aritméticas de 22 mil multiplicações e 25 mil somas ou subtrações para apenas 39 mil somas ou subtrações. Se analisadas as somas ou subtrações de um bit utilizadas, foi possível gerar um ganho superior a 59%. A partir dos algoritmos otimizados, foram desenvolvidas as arquiteturas, com foco em FPGAs Altera. Foram cinco as arquiteturas de DCTs 2-D desenvolvidas, uma para cada tamanho de transformada e uma capaz de processar todos os tamanhos de transformada. Em todos os casos as transformadas foram projetadas para usar a propriedade da separabilidade, ou seja, foram usadas duas instâncias de uma arquitetura de transformada 1-D e uma matriz de transposição. As transformadas 1-D foram desenvolvidas de forma puramente combinacional. Para todas as transformadas foi possível atingir taxas de processamento elevadas, capazes de suportar o processamento de vídeos de alta resolução.

Palavras-chave: Codificação de Vídeo. HEVC. DCT 2-D de Múltiplos Tamanhos. Otimização Algorítmica, Projeto em FPGA.

ABSTRACT

JESKE, Ricardo G. **Otimizações Algorítmicas e Desenvolvimento Arquitetural para as DCTs do HEVC.** 2013. 107f. Dissertação - Mestrado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

Video coding is an area in constant evolution, especially in function of its relevance for consumer electronics, since devices which are able to process digital videos are currently more and more common. The video coder efficiency improvement is function of the relation between bit-rate and video quality and, to improve the encoder efficiency, the video coding systems are using more and more complex coding tools. This causes a big impact in the coding process complexity. This work is focused in the coding tools defined in the new emerging standard called HEVC – *High Eficiency Video Coding*. More specifically, this work focuses in the variable size discrete cosine transforms defined in the HEVC. There are four 2-D DCT transforms sizes defined in the HEVC standard: 32x32, 16x16, 8x8 and 4x4. This is a novelty of this standard and its use increases the coder efficiency and complexity. This work present two main contributions: (a) the development of an algorithm to optimize the 2-D DCT algorithms (with the transformation of multiplications in shift-adds and through the common sub-expressions sharing) and the software implementation of this algorithm to generate the simplified 2-D DCT algorithms and (b) the hardware design of the optimized algorithms, focusing in low cost and high processing rates. These optimizations are especially necessary when battery powered devices are considered. The optimizations allowed, in the best case (32x32 DCT), an expressive reduction in the number of operations: from 22 thousand of multiplications and 25 thousand of additions or subtractions to 29 thousand of additions or subtractions. If the number of one-bit additions or subtractions is considered, the developed optimizations are able to generate gains higher than 59%. The architectures were designed using the simplified algorithms, focusing in Altera FPGAs. Five architectures were designed, one for each 2-D DCT size and one which is able to process all four sizes of the DCTs. In all cases, the transforms were designed to use the separabilitty property, then two instances of the 1-D DCT transforms were joined by a transposition matrix to form the 2-D DCT transforms. The 1-D transforms were designed in a fully combinational way. The final synthesis results showed that all designed 2-D DCT architectures are able to reach very high processing rates, which allow the processing of high definition videos in real time.

Keywords: Video Coding. HEVC. Multiple Size 2-D DCT. Algorithmic Optimization, FPGA Design.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

1-D	Uma Dimensão
2-D	Duas Dimensões
4:2:0	Método de amostragem: componentes de crominância têm metade da resolução horizontal e vertical de componentes de luminância.
4:2:2	Método de amostragem: componentes de crominância têm metade da resolução horizontal de componentes de luminância.
4:4:4	Método de amostragem: componentes de crominância têm a mesma resolução de componentes de luminância.
ALM	Módulos Lógicos Adaptativos
AMVP	*Adaptive Motion Vector Prediction*
AVC	*Advanced Video Coding*
BitDepth	Profundidade de *bits* utilizada nos cálculos.
BO	*Band Offset*
CABAC	*Context Adaptive Binary Arithmetic Coding*
CAVLC	*Context Adaptive Variable Length Coding*
Cb	*Chrominance Blue*
Cr	*Chrominance Red*
CTU	*Coding Tree Units*
CU	*Coding Units*
DC	*Direct Current*
DCT	*Discrete Cosine Transform*
DF	*Deblocking Filter*
DST	*Discrete Sine Transform*

DSP	*Digital Signal Processors*
EO	*Edge Offset*
EOS	Equações Otimizadas pelo Software
FHD	*Full High Definition*
FME	*Fractionary Motion Estimation*
FPGA	*Field-Programmable Gate Array*
GACI	Grupo de Arquiteturas e Circuitos Integrados
GB	Giga Byte
Gbps	Giga *bits* por segundo
H.264	Um padrão de codificação de vídeo
HD	*High Definition*
HEVC	*High Efficiency Video Coding*
HM	*HEVC Test Model*
HM4	Quarto *HEVC Test Model*
HM9	Nono *HEVC Test Model*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
ITU	*International Telecommunication Union*
ITU-T	Setor da ITU que coordena os padrões para as telecomunicações.
JCT – VC	*Joint Colaborative Team – Video Coding*
MCM	*Multiple Constant Multiplication*
MHz	Mega-Hertz, 10^6 hertz
MPEG-2	Um padrão de codificação multimídia
PU	*Prediction Units*
QFHD	*Quadruple Full High Definition*

Quadtree	Árvore em que um nó pai pode ser dividido em quatro nós filhos, sendo que um nó filho pode se tornar um nó pai e ser novamente dividido em quatro nós filhos.
RGB	*Red, Green, Blue*
RTL	*Register Transfer Level*
SAO	*Sample Adaptive Offset*
SD	*Standard Definition*
TU	*Transform Units*
Treeblocks	Um bloco NxN de amostras de luminância e dois blocos de amostras correspondentes de crominância de uma imagem que tem três matrizes de amostra.
UFPel	Universidade Federal de Pelotas
VHDL	*VHSIC Hardware Description Language*
Y	*Luminance*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, a resolução e a qualidade dos vídeos digitais têm sido ampliadas de forma rápida e contínua. Além disso, estes vídeos estão sendo utilizados em um número cada vez mais diversificado de dispositivos (*smartphones*, *set-top-boxes*, televisores digitais, *blu-ray players*, etc.). Por isso, o estudo e o aperfeiçoamento de codificadores/decodificadores de vídeos é uma atividade de extrema relevância no cenário atual, pois os diversos dispositivos que processam vídeos digitais, com suas diferentes características, devem ser capazes de processar vídeos de elevada resolução em tempo real. Assim, questões como taxa de compressão, qualidade do vídeo, tempo de processamento e consumo de energia devem ser foco de otimização nas investigações desta área.

A compressão de vídeos é essencial nas aplicações que manipulam vídeos digitais, pois um vídeo não comprimido utiliza um número proibitivo de *bits* para a sua representação (AGOSTINI, 2007). O H.264/AVC (ITU, 2005) é o padrão mais atual nesta área, apresentando ganhos significativos de compressão quando comparado ao padrão MPEG-2 (ITU, 1994). Em janeiro de 2010, foi criado o JCT-VC (*Joint Colaborative Team – Video Coding*), formado por especialistas da ITU-T e do ISO/IEC, para começar o desenvolvimento do novo padrão de codificação, chamado HEVC – *High Efficiency Video Coding* (JCT-VC, 2010). O objetivo do HEVC é aumentar em 50% a compressão dos vídeos em relação ao padrão H.264/AVC, mantendo a mesma qualidade de imagem. O HEVC ainda está em desenvolvimento, mas algumas importantes modificações em relação ao H.264/AVC já foram definidas.

Como os dispositivos atuais que manipulam vídeos digitais precisam ser capazes de processar vídeos de elevada resolução em tempo real, uma

solução muito usual é a implementação em hardware dos algoritmos que descrevem os cálculos efetuados no processo de codificação. Este procedimento é adotado tanto por possibilitar taxas de processamento elevadas, quanto por possibilitar um menor consumo de energia quando comparado a implementação em software.

Este trabalho aborda, especificamente, implementações em hardware para as transformadas DCT do HEVC, que visam baixo custo de hardware e elevada taxa de processamento. Foram desenvolvidas e implementadas heurísticas de otimização nos algoritmos das transformadas DCT visando a implementação em hardware. Estas técnicas de otimização utilizadas resultaram em uma grande redução nos recursos de hardware necessários para implementar a transformada e também conduziram a uma expressiva elevação na velocidade de processamento, quando comparado à implementação gerada com a transcrição direta do algoritmo extraído do HM. Estes ganhos ficam mais evidenciados nas DCTs de tamanhos maiores. A partir das otimizações algorítmicas, foram implementadas as transformadas DCT 1-D e DCT 2-D de diferentes tamanhos, 4x4, 8x8, 16x16 e 32x32 presentes no módulo das transformadas do padrão emergente de codificação de vídeos HEVC. Cada transformada foi implementada individualmente, assim como foi desenvolvida uma arquitetura Multi-DCT capaz de realizar a DCT de qualquer tamanho presente no HEVC.

Até o presente momento existem poucos trabalhos na literatura que abordam questões sobre o HEVC, uma vez que este ainda se encontra em fase de desenvolvimento. Alguns destes trabalhos já publicados estão relacionados com as transformadas DCT no HEVC, incluindo projetos de hardware, como (JESKE et al., 2012), (BUDAGAVI et al., 2012), (AHMED et al., 2011), (EDIRISURIYA et al., 2012), (CHENG et al., 2011), (MARTUZA et al, 2012) e (SHEN et al., 2012). Mas não foi encontrado nenhum trabalho além dos documentos JCT-VC relacionados com a redução de complexidade da DCT 2-D no HEVC. As publicações geradas a partir deste trabalho, como (JESKE et al., 2012), estão todas apresentadas no Apêndice B desta dissertação.

Como a codificação de vídeos tem ganhado cada vez mais destaque nos dispositivos atuais e como o HEVC é o estado da arte nesta área, são

importantes trabalhos, como esta dissertação, que focam nestes temas. A contribuição deste trabalho ficará mais evidente quando serão apresentados os resultados obtidos tanto nas otimizações algorítmicas quanto no projeto de hardware, que estão apresentados nos próximos capítulos.

1.1 Motivação

A definição do foco deste trabalho na implementação em hardware das transformadas DCT do HEVC foi função de uma soma de fatores, dentre eles os principais foram:

- a codificação de vídeos processa um volume muito elevado de dados e para processar os vídeos em tempo real, são exigidas elevadas taxas de processamento. Nesse contexto, implementações em hardware se mostram extremamente vantajosas quando comparadas com software rodando sobre processadores de propósito geral, devido as elevadas taxas de processamento e ao reduzido consumo energético que o hardware permite quando comparado a implementações em software;

- por ainda estar em fase de desenvolvimento, existem muito poucos trabalhos na literatura relatando o desenvolvimento de hardware para o HEVC, por isso o tema ainda está repleto de contribuições para serem dadas;

- o módulo das transformadas presente nos codificadores mostra-se um ponto de gargalo com um fluxo elevado de dados, sendo de grande relevância o desenvolvimento de soluções algorítmicas mais eficientes e do projeto de hardware dedicado capaz de atingir as elevadas taxas de processamento exigidas, com o menor consumo possível de energia.

1.2 Objetivos

O objetivo deste trabalho é apresentar arquiteturas em hardware para a transformada DCT do novo padrão de codificação de vídeo HEVC, bem como as otimizações algorítmicas desenvolvidas. As soluções propostas visam implementações de baixo consumo de hardware com altas taxas de processamento.

1.3 Organização do Trabalho

Este trabalho divide-se em sete capítulos, de forma a explicar todos os principais conceitos envolvidos na codificação de vídeo, bem como a descrição do novo padrão de codificação que está sendo desenvolvido (HEVC), as heurísticas de otimização utilizadas, assim como o software desenvolvido para realizar as otimizações e a implementação em hardware das transformadas DCTs.

No segundo capítulo, é feita uma descrição dos principais conceitos que envolvem o processo de codificação de vídeos. Já no terceiro capítulo, é feita uma descrição das ferramentas que compõem o novo padrão de codificação de vídeos HEVC.

Em seguida, no quarto capítulo, é abordado o módulo das transformadas, o qual, dentre todas as etapas que envolvem um codificador de vídeos, é o foco deste trabalho.

Já no quinto capítulo, são descritas as heurísticas de otimização, o algoritmo que foi desenvolvido com o intuito de realizar as otimizações de forma automatizada, bem como as novas equações geradas pelo software de otimização que correspondem às DCTs de diferentes tamanhos.

O sexto capítulo apresenta as implementações em hardware e análise dos resultados obtidos para a implementação em hardware das transformadas DCT.

No sétimo e último capítulo, são expostas as conclusões do trabalho através de um balanço entre os hardwares implementados e os resultados obtidos.

Por fim, no Apêndice A, são apresentadas as equações geradas pelo software de otimização e no Apêndice B uma relação com as publicações geradas a partir deste trabalho. Além disso, no Anexo A constam as matrizes de constantes de multiplicação utilizadas nos cálculos da DCT no HEVC e no Anexo B os algoritmos das transformadas DCT descritas no HEVC.

2 CODIFICADOR DE VÍDEO DIGITAL

Nesta seção, será apresentado o funcionamento de um codificador de vídeos genérico e mostrada a importância dessa ferramenta para aplicações que manipulam vídeos digitais.

2.1 Volume de Dados em Vídeos Digitais

Cada imagem que compõe um vídeo digital, também chamada de quadro, é formada por um determinado número de *pixels* ou pontos: quanto maior a resolução da imagem, proporcionalmente maior será o número de *pixels* que a compõe. Na Figura 2.1, estão apresentadas algumas das principais resoluções dos vídeos digitais atuais.

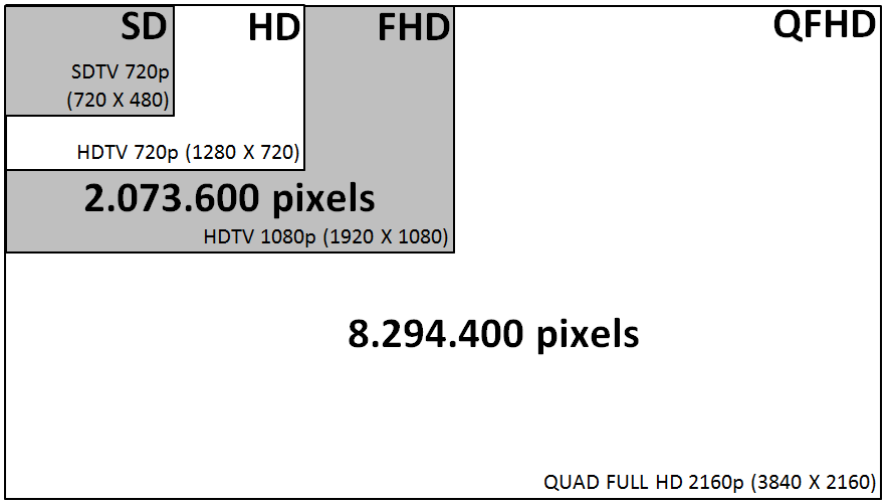

Figura 2.1 – Resolução dos Vídeos Digitais

Apenas para demonstrar a importante tarefa que os codificadores de vídeos exercem comprimindo os vídeos digitais, será exposto a seguir um exemplo numérico.

Tomando como exemplo um vídeo com resolução FHD (1920 x 1080 *pixels*) e considerando que são utilizados 24 *bits* para representar numericamente cada *pixel* (três amostras de cores de 8 bits por *pixel*), em cada quadro FHD existem 2.073.600 *pixels,* sendo assim, são necessários 49.766.400 *bits* para representá-lo.

Considerando uma taxa de processamento de 30 quadros por segundo, para reproduções em tempo real, neste caso para vídeos FHD sem compressão seriam necessários 1,49 Gbps no canal de comunicação. Um vídeo de 10 minutos de duração utilizaria um espaço de armazenamento de 112 GB.

Para vídeos com resolução QFHD (*Quad* FHD), que possuem um número de *pixels* quatro vezes maior que a resolução FHD, seriam necessários 448 GB de espaço em disco para armazenamento de um vídeo com 10 minutos de duração. Já a largura de banda para possibilitar transmissões em tempo real precisaria ser de 6 Gbps.

Estes valores mostram que se torna inviável implementar aplicações práticas reais que utilizam vídeos digitais sem técnicas de compressão.

2.2 Etapas de Codificação

Cada etapa presente em um codificador de vídeo tem como objetivo contribuir para a redução da redundância de informação presente nos dados que representam o vídeo. Desta forma, após todas as etapas que compõem o codificador, torna-se possível representar o mesmo vídeo com um número reduzido de *bits*.

Na Figura 2.2 é apresentado um diagrama das principais etapas que compõem um codificador de vídeo atual, que é compatível com o padrão H.264/AVC e também com o padrão emergente HEVC.

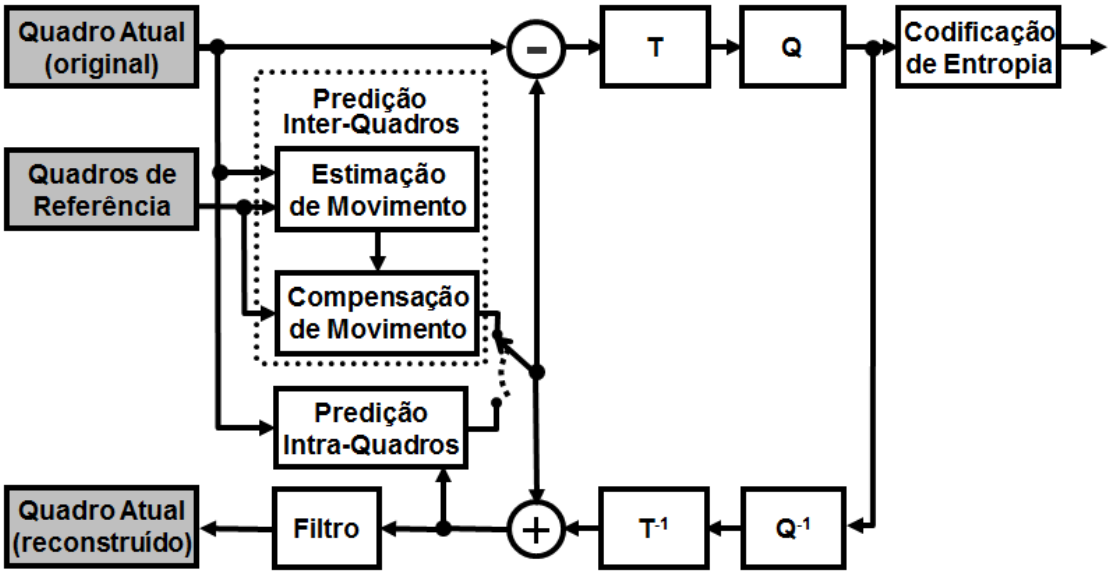

Figura 2.2 – Modelo de um codificador de vídeo

Cada quadro é dividido em blocos que podem assumir diferentes tamanhos. O tamanho de cada bloco depende do tipo de codificador e da decisão tomada pelo próprio codificador após analisar o quadro a ser codificado.

Inicialmente, é realizada a Codificação Inter-Quadros e Intra-Quadros para cada bloco e, após essas etapas, o codificador toma uma decisão de qual das duas apresenta melhores resultados. Então, é realizada uma subtração entre o bloco original e o bloco predito, gerando um resíduo.

A Codificação Inter-Quadros tem como objetivo eliminar a redundância temporal. Essa redundância de informações ocorre devido a taxa de amostragem e isso gera muitos blocos com dados redundantes de um quadro para outro em um vídeo. Além disso, alguns blocos de *pixels* sofrem apenas um deslocamento de um quadro para o outro. Um exemplo disso é um objeto rígido (indeformável) em movimento: os blocos que representam o objeto possuem valores similares, apenas se deslocando de posição a cada quadro do vídeo. Para realizar a predição Inter-Quadros é necessário que exista pelo menos um quadro previamente processado, para ser usado como referência. Os quadros gerados a partir de quadros anteriores são chamados de quadros tipo P ou tipo B (RICHARDSON, 2010).

Na Codificação Intra-Quadros, busca-se eliminar a redundância espacial. Essa redundância de informação é causada pelo fato de *pixels*

vizinhos tenderem a possuir valores similares e quanto maior a resolução do vídeo, mais acentua-se essa redundância. A predição Intra-Quadro é sempre realizada no primeiro quadro do vídeo, chamado quadro I, já que não depende que quadros previamente codificados. Com a função de sincronismo e de redução na propagação de erros, é comum a inserção de quadros I a intervalos constantes de quadros P ou B, e todos os blocos dentro de quadros I devem ser preditos com a predição Intra-Quadro. Além disso, também é possível usar a predição Intra-Quadro em quadros do tipo P ou B, já que, em alguns casos, a predição Intra-Quadro atinge resultados superiores à predição Inter-Quadros neste cenário (RICHARDSON, 2010).

Após a etapa de predição, o resíduo é enviado para o módulo das transformadas, que é responsável por transformar os dados do domínio espacial para o domínio das frequências. Neste domínio, é possível identificar as frequências menos relevantes para o sistema visual humano e eliminá-las através do processo de quantização. A quantização é realizada através de uma divisão inteira dos coeficientes transformados por constantes predefinidas e, portanto, é uma etapa que gera perdas de informação no processo de codificação. Mas estas perdas podem ser imperceptíveis ao sistema visual humano, dependendo da configuração do codificador.

Por fim, a codificação de entropia visa eliminar a redundância entrópica, que está relacionada com a forma de representação dos símbolos codificados. A codificação de entropia utiliza diferentes técnicas e algoritmos de compressão sem perdas para representar informações com um menor número de *bits*. (AGOSTINI, 2007).

3 PADRÃO HEVC

Nesta seção será feita uma descrição geral das ferramentas que compõem o novo padrão de codificação HEVC. As informações aqui contidas foram extraídas da versão do HEVC, HM9 - HEVC *Model* - (JCT-VC, 2012) e do Draft9 (JCT-VC, 2013). O nono modelo de testes do HEVC (HM9), foi definido por decisões tomadas na reunião do JCT-VC realizada em Shanghai, China, de 10 a 19 de outubro de 2012.

3.1 Particionamento de Quadro

O HEVC prevê diversas formas de particionamento do quadro para aumentar a eficiência da codificação, como: CTU, *slice*, *tile*, CU, PU e TU. A seguir será apresentado como é formado cada um destes diferentes tipos de divisões de quadros.

3.1.1 Particionamento CTU

No HEVC, cada quadro é dividido em unidades de codificação (CTUs - *Coding Tree Units*). A CTU consiste de um bloco NxN de amostras de luminância juntamente com os dois blocos de crominância. O conceito de CTU é amplamente análogo ao de macroblocos nos padrões anteriores, tal como no AVC (ITU, 2005). O tamanho máximo do bloco de luminância numa CTU é especificado em 64x64 no perfil principal (JCT-VC, 2012), (HAN, 2010).

3.1.2 Estruturas *Slice* e *Tile*

Um *slice* é composto por um número inteiro de CTUs adjacentes, formando um conjunto de corte independente. Os *slices* são projetados para serem decodificados de forma independente, por isso a predição não é feita nas bordas dos *slices*, e a codificação de entropia é reinicializada em cada *slice*.

Um *tile*, assim como um *slice*, também é formado por um número inteiro de CTUs*,* porém é especificado como sendo uma região retangular.

Um *tile* pode ser formado de blocos contidos em mais de um *slice*, e um *slice* pode possuir blocos contidos em mais de um *tile*. Entretanto, uma ou ambas das seguintes condições devem ser satisfeitas para cada *slice* e *tile*:

a) Todas as CTUs de um *slice* pertencem ao mesmo *tile*.

b) Todas as CTUs de um *tile* pertencem ao mesmo *slice.*

A Figura 3.1, mostra um exemplo onde um *frame* formado por 11x9 CTUs, é dividido horizontalmente formado 2 *tiles* e apresentando 3 *slices.*

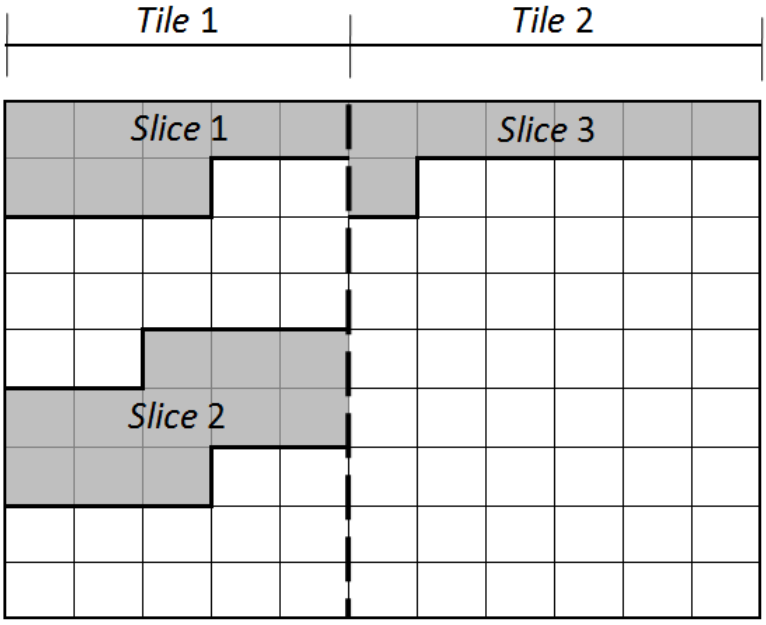

Figura 3.1 – Exemplo de *Tile* e *Slice*

3.1.3 Unidade de Codificação (CU)

Cada CTU é composta por uma ou mais unidades básicas de codificação, chamadas *Coding Units* (CU). A CU pode ser dividida recursivamente em quatro blocos de tamanhos iguais a partir da CTU e chegando a um tamanho mínimo de 8x8 amostras. Este processo recursivo forma uma árvore quaternária ou *quadtree* composta por blocos de CU, assumindo dimensões que variam desde 8x8 *pixels* até o tamanho da própria CTU, ou seja, no máximo 64x64 *pixels*. A Figura 3.2 apresenta um possível particionamento da CTU em uma *quadtree* de CUs.

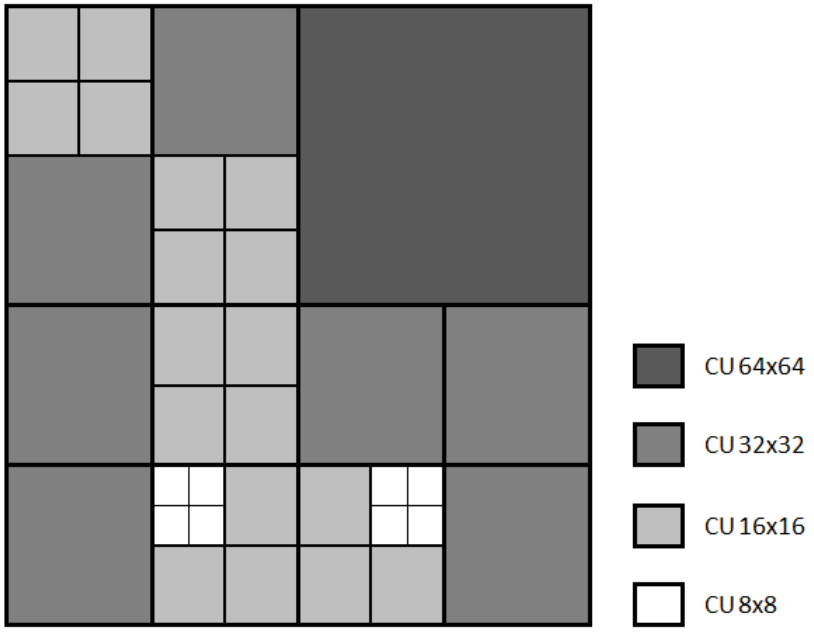

Figura 3.2 – Exemplo de Particionamento da CTU – Vista planar

Já na Figura 3.3 é apresentado o mesmo particionamento só que em uma vista 3D.

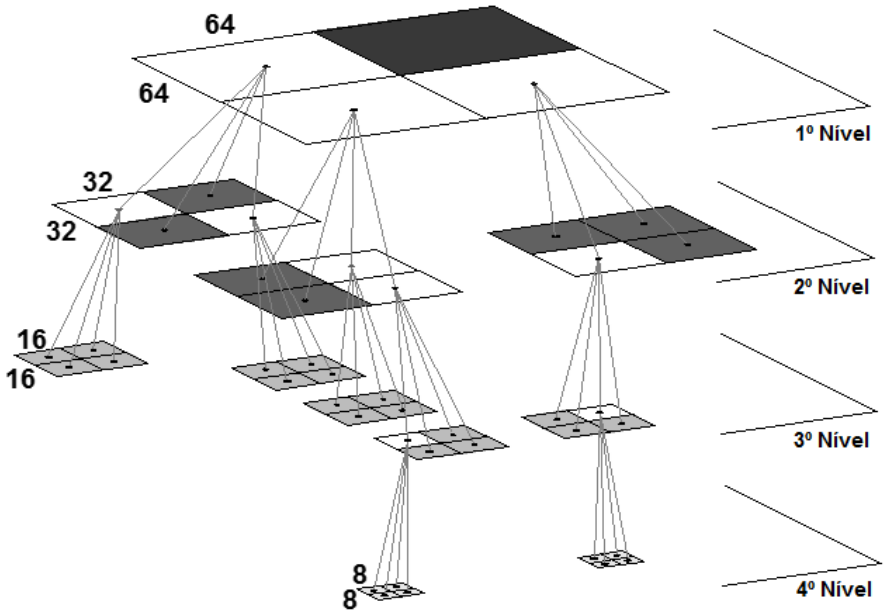

Figura 3.3 – Exemplo de Particionamento da CTU – Vista 3D

Uma CU forma uma região que compartilha o mesmo modo de predição (inter ou intra).

3.1.4 Unidade de Predição (PU)

A Unidade de Predição, chamada de *Prediction Unit (*PU) é a unidade básica utilizada para o transporte da informação relacionada com os processos de predição. Em geral, não são restritas a estarem na forma quadrada, para facilitar a partição que corresponde a limites de objetos reais na imagem. Cada CU pode ser dividida em uma ou mais PUs, conforme os formatos exibidos na Figura 3.4, cada um dos quais podem ser tão grandes como a própia CU (2Nx2N) ou tão pequenos quanto o tamanho do bloco 8x4 ou 4x8 em luminância.

A Figura 3.4 ilustra os 8 modos de partição para blocos CUs gerados pela predição inter-quadros, onde N representa metade do tamanho da CU.

Para blocos gerados pela predição intra-quados, somente são permitidos blocos quadrados (2Nx2N e NxN). A divisão do tamanho NxN também só é permitida quando o bloco correspondente de CU possuir sua dimensão mínima (8x8 *pixels*).

A predição inter-quadros permite PUs quadradas ou retangulares, a fim de facilitar o casamento com bordas de objetos nas imagens. As PUs podem assumir tamanhos 2Nx2N, 2NxN, Nx2N e NxN, sendo o menor tamanho permitido 8x4 ou 4x8 amostras de luminância. A divisão NxN e as divisões assimétricas só são permitidas para CUs maiores do que 8x8 (JCT-VC, 2012).

A divisão 2Nx2N é composta por uma PU, a NxN possui quatro PUs e todas as demais partições possuem duas PUs.

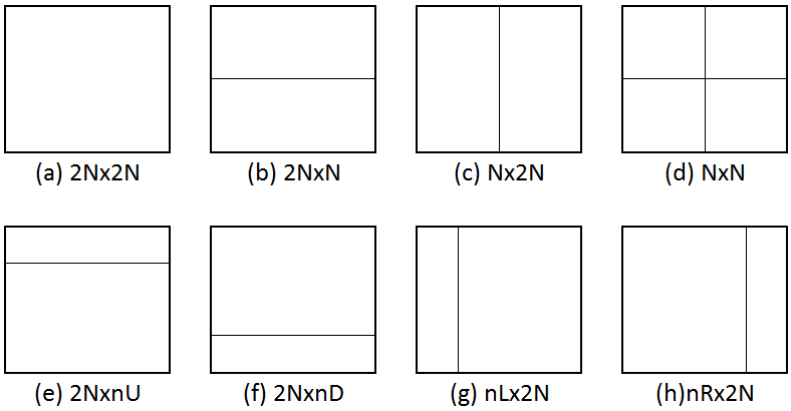

Figura 3.4 – Oito tipos de estrutura de divisão de CU em PUs

3.1.5 Unidade de Transformada (TU)

No HEVC, as unidades básicas para o processo de transformada e quantização são chamadas de *Transform Units* (TU). O seu formato é sempre quadrado e pode assumir um tamanho de 4x4, 8x8, 16x16 e 32x32 amostras. Do mesmo modo que as CUs, as TUs são estruturadas em *quadtrees*. Cada CU pode conter uma ou mais TUs. A Figura 3.5 ilustra uma CU de 32x32 amostras dividida em TUs de dimensões 16x16, 8x8 e 4x4. A profundidade máxima *quadtree* é ajustável. As TUs transportam para o módulo das transformadas os dados resultantes do processamento das PUs, ou seja, os resíduos gerados pela predição intra-quadro ou inter-quadros.

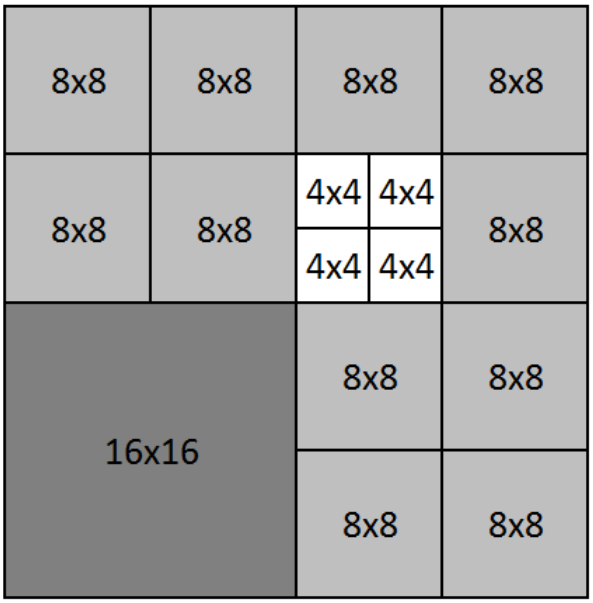

Figura 3.5 – Exemplo de PU 32x32 dividida em TUs

Para CUs geradas pela predição inter-quadros, a TU pode possuir um tamanho maior que a PU, o que significa que a TU pode conter internamente limites (bordas) de PUs. No entanto, isso não ocorre para CUs geradas pela predição intra-quadros: neste caso a TU não pode cruzar os limites de uma PU.

4 TRANSFORMADAS NO HEVC

Dentre todas as etapas que compõem o codificador HEVC, o módulo das transformadas é importante pois é nesta etapa que os dados são convertidos do domínio espacial para o domínio das frequências, ampliando a eficiência da quantização que, por sua vez amplia a eficiência da codificação de entropia, elevando muito a taxa de compressão.

Normalmente, utilizando a Transformada Discreta dos Cossenos (DCT), a etapa de transformadas tem a finalidade de concentrar a energia da imagem em apenas poucos coeficientes numéricos. Deste modo, as etapas seguintes (quantização e codificação de entropia) podem ser aplicadas de forma muito mais eficiente.

Para um melhor entendimento do objetivo do módulo das transformadas, é possível tomar como exemplo a Figura 4.1, que está representando uma matriz de tamanho 8x8 composta por resíduos provenientes da diferença entre o quadro de referência e o quadro atual, gerados nas etapas anteriores do codificador.

Como pode ser percebido na matriz de entrada (Figura 4.1), toda a matriz está preenchida com valores dos resíduos, e após os cálculos que compõem a DCT, estes mesmos valores são transformados do domínio espacial para o domínio das frequências e agora, devido as características dos cálculos, os valores ficam concentrados no canto superior esquerdo de uma matriz de saída(Figura 4.2).

A matriz, após passar pela DCT 2-D, apresenta os valores como o exemplo da Figura 4.2. Na sequência, estes dados passam pela etapa de quantização e ficam no formato apresentado na Figura 4.3.

-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-73	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34

Figura 4.1 – Exemplo de matriz de entrada da DCT 8x8

-415	-30	-61	27	56	-20	-2	0
4	-22	-61	10	13	-7	-9	5
-47	7	77	-25	-29	10	5	-6
-49	12	34	-15	-10	6	2	2
12	-7	-13	-4	-2	2	-3	3
-8	3	2	-6	-2	1	4	2
-1	0	0	-2	-1	-3	4	-1
0	0	-1	-4	-1	0	1	2

Figura 4.2 - Exemplo de matriz de saída da DCT 8x8

-26	-3	-6	2	2	-1	0	0
0	-2	-4	1	1	0	0	0
-3	1	5	-1	-1	0	0	0
-4	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figura 4.3 – Exemplo de matriz de saída da quantização

Como pode ser percebido na Figura 4.3, agora o mesmo bloco de 8x8 *pixels* está representado com apenas algumas posições diferentes de zero (é uma matriz esparsa) e todos estes valores não zero estão concentrados na região superior esquerda da matriz. Desta forma, o codificador de entropia, através de uma leitura em zigue-zague, será capaz de comprimir com grande eficiência esta matriz gerada pela quantização.

No processo de codificação, os cálculos da transformada DCT para um determinado bloco, são executados mais de um vez, já que as transformadas

estão incluídas no *loop* de codificação. Assim, um bloco precisa ser avaliado de acordo com as várias possibilidades de predição intra-quadro e inter-quadros antes que seja definida a predição escolhida e as transformadas fazem parte desse *loop*.

4.1 Transformada DCT

Uma transformada DCT genérica 2-D pode ser definida pelas equações (8) e (9), onde N e M são o número de pontos da DCT, F (u, v) é a entrada para a posição (u, v) da matriz de entrada, e *f*(i, j) é o coeficiente de saída.

$$F_{(u,v)} = \sqrt{\frac{2}{N}}.\sqrt{\frac{2}{M}}.\sum_{n=0}^{N-1}\sum_{m=0}^{M-1}\Lambda_{(i)}.\Lambda_{(j)}.\cos\left[\frac{\pi.u}{2.N}(2i+1)\right].\cos\left[\frac{\pi.v}{2.M}(2j+1)\right].f_{(i,j)} \quad (8)$$

$$\Lambda_{(k)} = \begin{cases} \frac{1}{\sqrt{2}} & \text{se k} = 0 \\ 1 & \text{outros} \end{cases} \quad (9)$$

A aplicação direta das equações (8) e (9), sem qualquer simplificação, usará um total de multiplicações e adições conforme a Tabela 4.1. Dado o número elevado de operações, é importante realizar investigações na tentativa de reduzir o número de cálculos necessários e reduzir a complexidade destes cálculos.

Tabela 4.1 – Número de operações matemáticas da DCT

DCT	NxM	Multiplicações	Adições
4x4	16	256	240
8x8	64	4.096	4.032
16x16	256	65.536	65.280
32x32	1.024	1.048.576	1.047.552

4.2 Transfomadas DCT no HEVC

A abordagem tradicional para reduzir a complexidade da DCT 2-D é o uso da propriedade da separabilidade (GHANBARI, 2003). A separabilidade considera que duas DCTs 1-D podem ser aplicadas para gerar a DCT 2-D. Por

conseguinte, no HEVC, a DCT 2-D é composta por dois passos subsequentes de uma transformada DCT 1-D, ligadas por uma etapa de transposição.

As transformadas DCT 1-D no HEVC apresentam um formato bem peculiar, em que a DCT de 4 pontos (usada para gerar a DCT 4x4) é parte integrante da DCT de 8 pontos (usada para gerar a DCT 8x8); por sua vez, a DCT de 8 pontos também é parte integrante da DCT de 16 pontos e isso se repete na DCT de 32 pontos. A Figura 4.4 apresenta de que forma as DCTs de diferentes tamanhos estão interligadas.

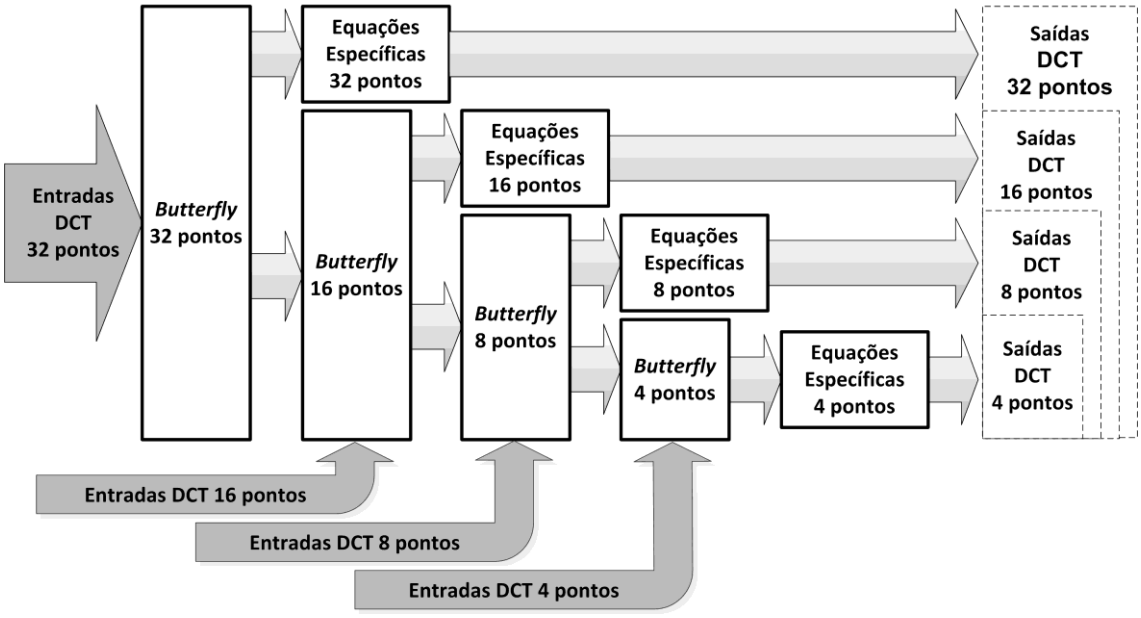

Figura 4.4 – Transformadas DCT 1-D no HEVC

Os primeiros cálculos realizados pela DCT 1-D seguem um padrão e sua representação lembra as asas de uma borboleta, por isso estes cálculos iniciais são popularmente chamados de *butterfly*.

Os dados de entrada passam, inicialmente, pelo bloco *butterfly*, onde são realizadas operações simples de somas e subtrações. Os resultados das subtrações são, então, passados como entrada para o bloco Equações, onde são realizados cálculos de equações específicas para cada DCT.

Já o resultado das operações de soma do bloco *butterfly* são passados como dados de entrada para o bloco *butterfly* de tamanho imediatamente inferior. Isto só não ocorre na DCT 4x4 onde todos os resultados do bloco *butterfly* (somas e subtrações) são utilizados nas equações.

Considerando a separabilidade, uma linha (ou coluna) da matriz de entrada é processada em cada etapa dos cálculos de cada transformada. Por isso, o número de entradas e saídas de cada DCT 1-D é o mesmo do seu tamanho, ou seja, a DCT de 32 pontos possui 32 entradas e 32 saídas.

A DCT de 4 pontos possui 4 equações específicas. Já na DCT de 8 pontos, das 8 equações que irão gerar suas saídas, 4 são reaproveitadas da DCT de 4 pontos, desta forma, apenas 4 novas equações são necessárias. Na DCT de 16 pontos, apenas 8 novas equações são utilizadas, pois as outras 8 saídas são reaproveitadas da DCT de 8 pontos. Por fim, a DCT de 32 pontos utiliza 16 novas equações, reaproveitando 16 equações da DCT de 16 pontos.

Nas subseções seguintes serão apresentados detalhadamente os cálculos internos de cada bloco apresentado na Figura 4.4.

A transformada DCT 2-D é composta por duas etapas subsequentes de transformadas DCT 1-D, sendo que a entrada da segunda transformada é a saída transposta da primeira transformada. Inicialmente, a 1ª DCT 1-D efetua a leitura linha a linha dos dados contidos em uma matriz de entrada e, após uma série de cálculos, os resultados são armazenados coluna a coluna em uma matriz de transposição de mesma dimensão. Após completado o processo de preenchimento de todos os resultados na matriz de transposição, a 2ª DCT 1-D é inicializada executando o mesmo procedimento anterior, fazendo a leitura dos dados linha a linha na matriz de transposição e, após executar os cálculos, os resultados são armazenados na matriz de saída. Este procedimento está ilustrado na Figura 4.5.

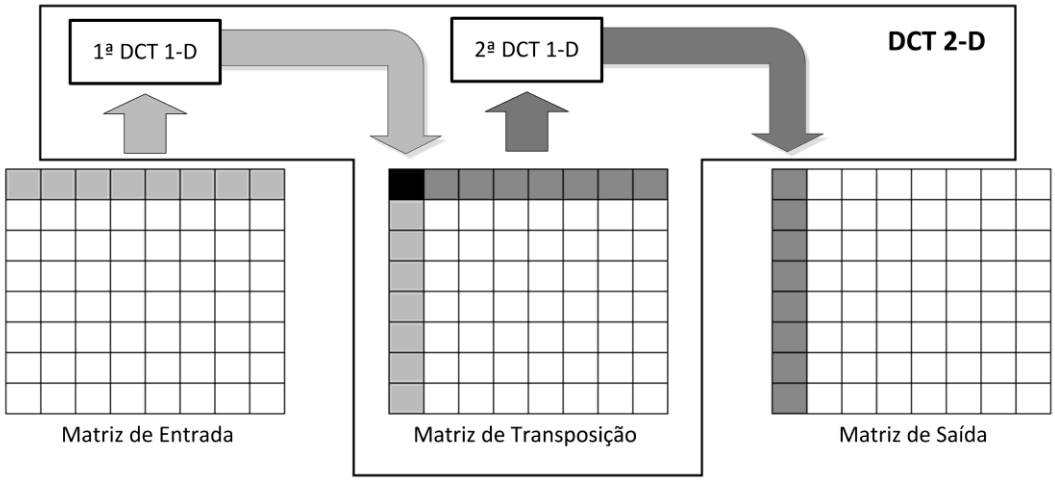

Figura 4.5 – Esquemático da DCT 2-D usando separabilidade.

No Anexo B, estão descritos os algoritmos das transformadas DCT 1-D, de tamanhos 4x4, 8x8, 16x16 e 32x32. Estes algoritmos utilizam as matrizes de constantes *g_ai[x][x],* conforme descrito no Anexo A. O algoritmo para a transformada da 1ª DCT 1-D é o mesmo utilizado na transformada da 2ª DCT 1-D, sendo que as únicas alterações são os valores das variáveis *add* e *shift*, usadas no arredondamento. Estas variáveis assumem os valores conforme descrito na Tabela 4.2.

Tabela 4.2 – Valores de *add* e *shift* das DCTs

DCT	1ª DCT 1-D		2ª DCT 1-D	
	add	***shift***	***add***	***shift***
4x4	1	1	128	8
8x8	2	2	256	9
16x16	4	3	512	10
32x32	8	4	1024	11

A seguir serão apresentados os cálculos de cada transformada DCT conforme o software de referência do HEVC.

4.2.1 DCT 4x4

A Figura 4.6 mostra os principais blocos da DCT de 4 pontos. Como ilustrado na Figura 4.6, os dados de entrada da DCT de 4 pontos passam inicialmente pelo bloco ‘butterfly 4 pontos’.

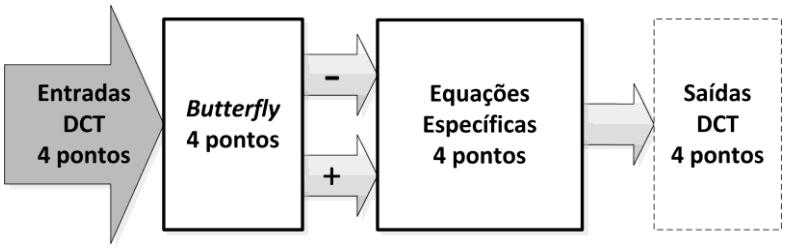

Figura 4.6 – DCT de 4 pontos

Os cálculos internos do bloco *butterfly* da transformada de 4 pontos estão ilustrados na Figura 4.7, assim como descritos na Tabela 4.3.

Na Figura 4.7 os W_n representam as entradas da DCT e é realizada uma operação de soma e uma de subtração com cada W_n seguindo o padrão da Figura 4.7. Estes cálculos iniciais são identificados neste trabalho como etapa a_n.

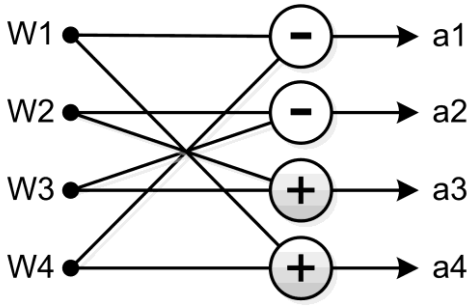

Figura 4.7 – *Butterfly* da DCT de 4 pontos

Tabela 4.3 – *Butterfly* da DCT de 4 pontos

Etapa a_n da DCT de 4 pontos - *butterfly*			
a1 = w1 - w4	a2 = w2 - w3	a3 = w2 + w3	a4 = w1 + w4

A Tabela 4.4 apresenta as 4 equações específicas que compõem a DCT de 4 pontos. Os valores de a_n são obtidos com os cálculos da Tabela 4.3 e os valores de *add* e *shift* foram apresentados na Tabela 4.2.

Tabela 4.4 – Equações da DCT de 4 pontos

Equações da DCT de 4 pontos		
saída_1	=	(64*a4 + 64*a3 + add) >> shift
saída_2	=	(83*a1 + 36*a2 + add) >> shift
saída_3	=	(64*a4 – 64*a3 + add) >> shift
saída_4	=	(36*a1 – 83*a2 + add) >> shift

Além das operações presentes na Tabela 4.4, ainda existem as 4 operações do bloco *butterfly*, apresentados na Tabela 4.3. Então, os cálculos da DCT de 4 pontos são efetuados utilizando-se 8 multiplicações e 12 somas ou subtrações. Como as transformadas DCT 2-D trabalham com matrizes quadradas, para completar todos os cálculos da DCT 2-D, deve-se executar os cálculos da DCT 1-D para cada linha da matriz de entrada. No caso da DCT 4x4, serão executados os cálculos da DCT de 4 pontos 4 vezes para gerar a

saída 4x4 da primeira DCT 1-D. Depois da transposição, os mesmos cálculos são realizados novamente pela segunda DCT 1-D, para gerar os resultados da DCT 2-D. Assim, a DCT 2-D usa duas vezes mais cálculos do que a DCT 1-D para processar a matriz de entrada.

O total de operações utilizadas pela DCT 4x4, de acordo com a definição do padrão HEVC, está descrito na tabela Tabela 4.5.

Tabela 4.5 – Número de operações matemáticas da DCT 4x4 no HEVC

DCT 4x4	Multiplicações	Adições
1-D	32	48
2-D	64	96

4.2.2 DCT 8x8

A Figura 4.8 representa a DCT de 8 pontos definida pelo HEVC. A DCT de 4 pontos é parte integrante da DCT de 8 pontos, como já explicado anteriormente e como pode-se notar na Figura 4.8.

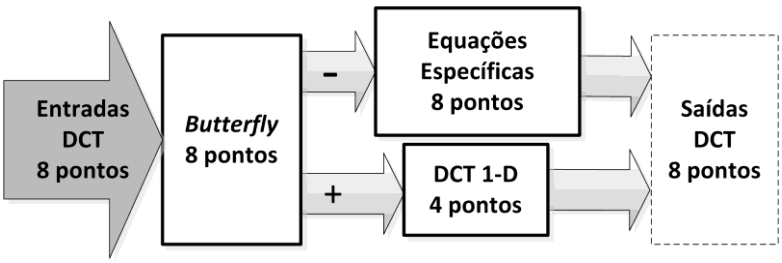

Figura 4.8 – DCT de 8 pontos

A Figura 4.9 descreve as operações internas do bloco *butterfly* de 8 pontos, que também estão descritas na Tabela 4.6.

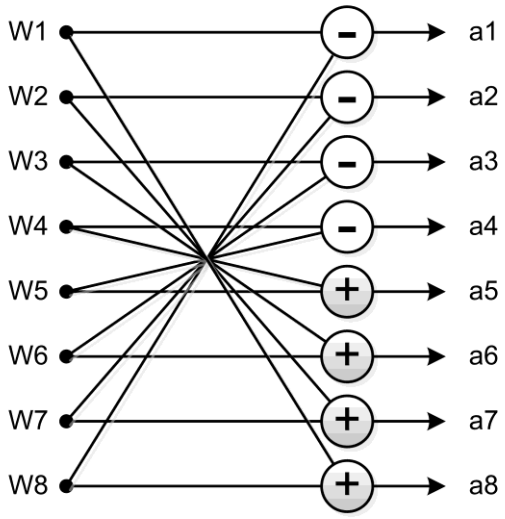

Figura 4.9 – *Butterfly* da DCT de 8 pontos

Tabela 4.6 - *Butterfly* da DCT de 8 pontos

Etapa a_n da DCT de 8 pontos - *butterfly*			
a1 = w1 - w8	a3 = w3 - w6	a5 = w4 + w5	a7 = w2 + w7
a2 = w2 - w7	a4 = w4 - w5	a6 = w3 + w6	a8 = w1 + w8

As equações que compõem a DCT de 8 pontos estão descritas na Tabela 4.7. Nesta tabela constam as 4 novas equações que fazem parte da DCT de 8 pontos. Como as outras 4 saídas são geradas pela DCT de 4 pontos, estas 4 equações restantes são as mesmas descritas na Tabela 4.4. Novamente, os valores de *add* e *shift* foram apresentados na Tabela 4.2.

Tabela 4.7 – Equações da DCT de 8 pontos

Equações da DCT de 8 pontos		
saída_2	=	(89*a1 + 75*a2 + 50*a3 + 18*a4 + add) >> shift
saída_4	=	(75*a1 - 18*a2 - 89*a3 - 50*a4 + add) >> shift
saída_6	=	(50*a1 - 89*a2 + 18*a3 + 75*a4 + add) >> shift
saída_8	=	(18*a1 - 50*a2 + 75*a3 - 89*a4 + add) >> shift
demais saídas	=	DCT de 4 pontos

O total de operações utilizadas por esta DCT, segundo definido pelo padrão HEVC, está descrito na

Tabela 4.8. Estes resultados consideram as mesmas observações discutidas na elaboração da Tabela 4.5. Comparando a Tabela 4.5 com a

Tabela 4.8, é possível perceber um aumento de seis vezes no número de cálculos realizados.

Tabela 4.8 – Número de operações matemáticas da DCT 8x8 no HEVC

DCT 8x8	Multiplicações	Adições
1-D	192	288
2-D	384	576

4.2.3 DCT 16x16

A Figura 4.10 mostra a estrutura da DCT de 16 pontos, seguindo a mesma lógica já discutida para a DCT de 8 pontos. Na Figura 4.10 está destacado o reaproveitamento dos resultados da DCT de 8 pontos apresentada na seção anterior (que, por sua vez, reaproveitava os resultados da DCT de 4 pontos).

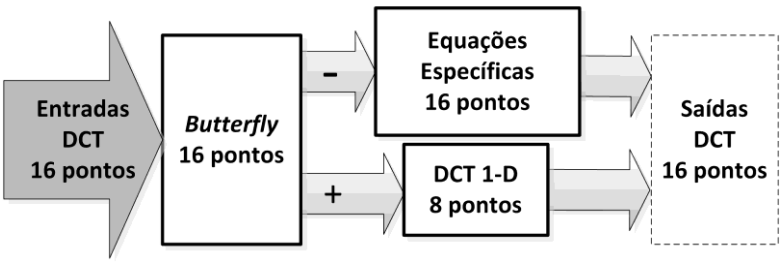

Figura 4.10 – DCT de 16 pontos

A Figura 4.11 descreve as operações do bloco *butterfly* da DCT de 16 pontos, também apresentando na Tabela 4.9.

As 8 novas equações específicas da DCT de 16 pontos estão descritas na Tabela 4.10. As outras 8 equações são reaproveitadas da DCT de 8 pontos. Os valores de *add* e *shift* são os mesmos apresentados na Tabela 4.2.

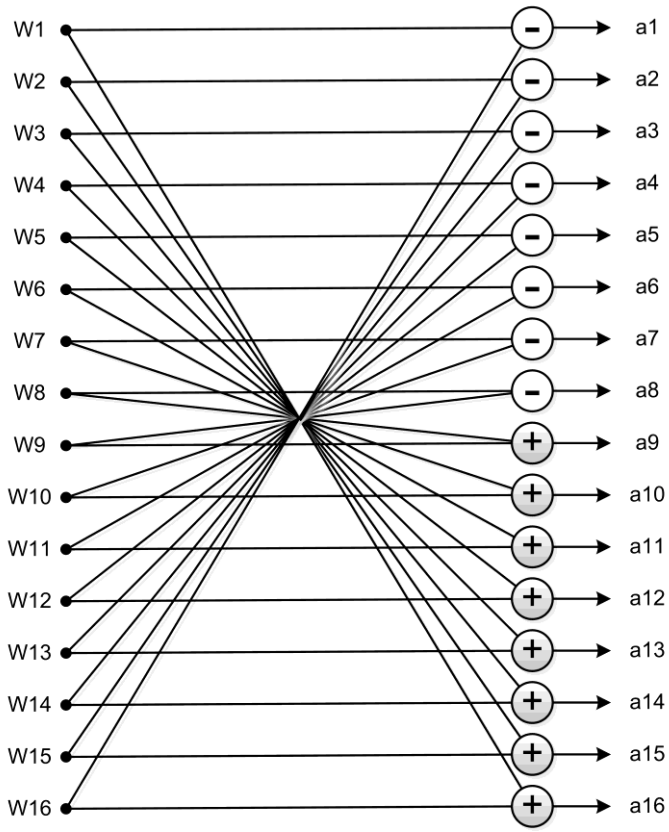

Figura 4.11 – *Butterfly* da DCT de 16 pontos

Tabela 4.9 - *Butterfly* da DCT de 16 pontos

Etapa a_n da DCT de 16 pontos			
a1 = w1 - w16	a5 = w5 - w12	a9 = w8 + w9	a13 = w4 + w13
a2 = w2 - w15	a6 = w6 - w11	a10 = w7 + w10	a14 = w3 + w14
a3 = w3 - w14	a7 = w7 - w10	a11 = w6 + w11	a15 = w2 + w15
a4 = w4 - w13	a8 = w8 - w9	a12 = w5 + w12	a16 = w1 + w16

Tabela 4.10 – Equações da DCT de 16 pontos

Equações da DCT de 16 pontos		
saída_2	=	(90*a1 + 87*a2 + 80*a3 + 70*a4 + 57*a5 + 43*a6 + 25*a7 + 9*a8 + add) >> shift
saída_4	=	(87*a1 + 57*a2 + 9*a3 - 43*a4 - 80*a5 - 90*a6 - 70*a7 - 25*a8 + add) >> shift
saída_6	=	(80*a1 + 9*a2 - 70*a3 - 87*a4 - 25*a5 + 57*a6 + 90*a7 + 43*a8 + add) >> shift
saída_8	=	(70*a1 - 43*a2 - 87*a3 + 9*a4 + 90*a5 + 25*a6 - 80*a7 - 57*a8 + add) >> shift
saída_10	=	(57*a1 - 80*a2 - 25*a3 + 90*a4 - 9*a5 - 87*a6 + 43*a7 + 70*a8 + add) >> shift
saída_12	=	(43*a1 - 90*a2 + 57*a3 + 25*a4 - 87*a5 + 70*a6 + 9*a7 - 80*a8 + add) >> shift
saída_14	=	(25*a1 - 70*a2 + 90*a3 - 80*a4 + 43*a5 + 9*a6 - 57*a7 + 87*a8 + add) >> shift
saída_16	=	(9*a1 - 25*a2 + 43*a3 - 57*a4 + 70*a5 - 80*a6 + 87*a7 - 90*a8 + add) >> shift
demais saídas	=	DCT de 8 pontos

A Tabela 4.11 mostra o total de operações da DCT de 16 pontos. Novamente, é possível ver o significativo aumento no número de cálculos quando comparados àqueles apresentados na Tabela 4.5 e na

Tabela 4.8. Em relação à Tabela 4.5, o aumento no número de cálculos é superior a 38 vezes, enquanto que em relação à

Tabela 4.8 a ampliação no número de cálculos é superior a seis vezes.

Tabela 4.11 – Número de operações matemáticas da DCT 16x16 no HEVC

DCT 16x16	Multiplicações	Adições
1-D	1.408	1.856
2-D	2.816	3.712

4.2.4 DCT 32x32

A Figura 4.12 apresenta a estrutura da DCT de 32 pontos, onde pode-se notar o reaproveitamento da DCT de 16 pontos.

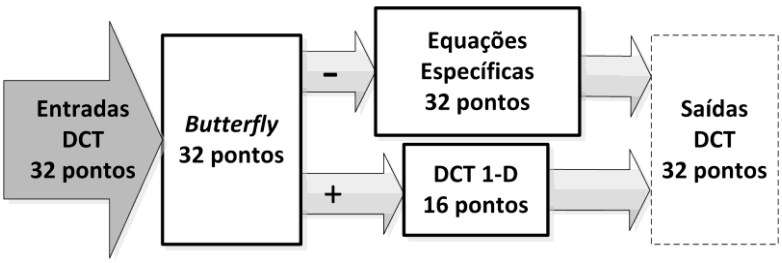

Figura 4.12 – DCT de 32 pontos

Tabela 4.12 - *Butterfly* da DCT de 32 pontos

Etapa a_n da DCT de 32 pontos			
a1 = w1 - w32	a9 = w9 - w24	a17 = w16 + w17	a25 = w8 + w25
a2 = w2 - w31	a10 = w10 - w23	a18 = w15 + w18	a26 = w7 + w26
a3 = w3 - w30	a11 = w11 - w22	a19 = w14 + w19	a27 = w6 + w27
a4 = w4 - w29	a12 = w12 - w21	a20 = w13 + w20	a28 = w5 + w28
a5 = w5 - w28	a13 = w13 - w20	a21 = w12 + w21	a29 = w4 + w29
a6 = w6 - w27	a14 = w14 - w19	a22 = w11 + w22	a30 = w3 + w30
a7 = w7 - w26	a15 = w15 - w18	a23 = w10 + w23	a31 = w2 + w31
a8 = w8 - w25	a16 = w16 - w17	a24 = w9 + w24	a32 = w1 + w32

A Tabela 4.12 apresenta os cálculos da etapa de *butterfly* da DCT de 32 pontos. A figura da *butterfly,* tal qual apresentada nas seções anteriores, foi omitida nesta seção por conta de seu excessivo tamanho.

A Tabela 4.13 apresenta as equações específicas da DCT de 32 pontos, onde é possível perceber a ampliação da complexidade do cálculo envolvido. Os valores de *add* e *shift* são apresentados na Tabela 4.2.

Tabela 4.13 – Equações da DCT de 32 pontos

Equações da DCT de 32 pontos		
saída_2	=	(90*a1 + 90*a2 + 88*a3 + 85*a4 + 82*a5 + 78*a6 + 73*a7 + 67*a8 + 61*a9 + 54*a10 + 46*a11 + 38*a12 + 31*a13 + 22*a14 + 13*a15 + 4*a16 + add) >> shift
saída_4	=	(90*a1 + 82*a2 + 67*a3 + 46*a4 + 22*a5 - 4*a6 - 31*a7 - 54*a8 - 73*a9 - 85*a10 - 90*a11 - 88*a12 - 78*a13 - 61*a14 - 38*a15 - 13*a16 + add) >> shift
saída_6	=	(88*a1 + 67*a2 + 31*a3 - 13*a4 - 54*a5 - 82*a6 - 90*a7 - 78*a8 - 46*a9 - 4*a10 + 38*a11 + 73*a12 + 90*a13 + 85*a14 + 61*a15 + 22*a16 + add) >> shift
saída_8	=	(85*a1 + 46*a2 - 13*a3 - 67*a4 - 90*a5 - 73*a6 - 22*a7 + 38*a8 + 82*a9 + 88*a10 + 54*a11 - 4*a12 - 61*a13 - 90*a14 - 78*a15 - 31*a16 + add) >> shift
saída_10	=	(82*a1 + 22*a2 - 54*a3 - 90*a4 - 61*a5 + 13*a6 + 78*a7 + 85*a8 + 31*a9 - 46*a10- 90*a11 - 67*a12 + 4*a13 + 73*a14 + 88*a15 + 38*a16 + add) >> shift
saída_12	=	(78*a1 - 4*a2 - 82*a3 - 73*a4 + 13*a5 + 85*a6 + 67*a7 - 22*a8 - 88*a9 - 61*a10 + 31*a11 + 90*a12 + 54*a13 - 38*a14 - 90*a15 - 46*a16 + add) >> shift
saída_14	=	(73*a1 - 31*a2 - 90*a3 - 22*a4 + 78*a5 + 67*a6 - 38*a7 - 90*a8 - 13*a9 + 82*a10 + 61*a11 - 46*a12 - 88*a13 - 4*a14 + 85*a15 + 54*a16 + add) >> shift
saída_16	=	(67*a1 - 54*a2 - 78*a3 + 38*a4 + 85*a5 - 22*a6 - 90*a7 + 4*a8 + 90*a9 + 13*10 - 88*a11 - 31*a12 + 82*a13 + 46*a14 - 73*a15 - 61*a16 + add) >> shift
saída_18	=	(61*a1 - 73*a2 - 46*a3 + 82*a4 + 31*a5 - 88*a6 - 13*a7 + 90*a8 - 4*a9 - 90*a10 + 22*a11 + 85*a12 - 38*a13 - 78*a14 + 54*a15 + 67*a16 + add) >> shift
saída_20	=	(54*a1 - 85*a2 - 4*a3 + 88*a4 - 46*a5 - 61*a6 + 82*a7 + 13*a8 - 90*a9 + 38*a10 + 67*a11 - 78*a12 - 22*a13 + 90*a14 - 31*a15 - 73*a16 + add) >> shift
saída_22	=	(46*a1 - 90*a2 + 38*a3 + 54*a4 - 90*a5 + 31*a6 + 61*a7 - 88*a8 + 22*a9 + 67*a10 - 85*a11 + 13*a12 + 73*a13 - 82*a14 + 4*a15 + 78*a16 + add) >> shift
saída_24	=	(38*a1 - 88*a2 + 73*a3 - 4*a4 - 67*a5 + 90*a6 - 46*a7 - 31*a8 + 85*a9 - 78*a10 + 13*a11 + 61*a12 - 90*a13 + 54*a14 + 22*a15 - 82*a16 + add) >> shift
saída_26	=	(31*a1 - 78*a2 + 90*a3 - 61*a4 + 4*a5 + 54*a6 - 88*a7 + 82*a8 - 38*a9 - 22*a10 + 73*a11 - 90*a12 + 67*a13 - 13*a14 - 46*a15 + 85*a16 + add) >> shift
saída_28	=	(22*a1 - 61*a2 + 85*a3 - 90*a4 + 73*a5 - 38*a6 - 4*a7 + 46*a8 - 78*a9 + 90*a10 - 82*a11 + 54*a12 - 13*a13 - 31*a14 + 67*a15 - 88*a16 + add) >> shift
saída_30	=	(13*a1 - 38*a2 + 61*a3 - 78*a4 + 88*a5 - 90*a6 + 85*a7 - 73*a8 + 54*a9 - 31*a10 + 4*a11 + 22*a12 - 46*a13 + 67*a14 - 82*a15 + 90*a16 + add) >> shift
saída_32	=	(4*a1 - 13*a2 + 22*a3 - 31*a4 + 38*a5 - 46*a6 + 54*a7 - 61*a8 + 67*a9 - 73*a10 + 78*a11 - 82*a12 + 85*a13 - 88*a14 + 90*a15 - 90*a16 + add) >> shift
Demais Saídas	=	DCT de 16 pontos

Na

Tabela 4.14 apresenta o número total de operações da DCT de 32 pontos. Com o aumento no tamanho da transformada, o número de cálculos

cresce rapidamente. Se comparados com os valores apresentados na Tabela 4.5 (DCT 4x4), na Tabela 4.8 (DCT 8x8) e na

Tabela 4.11 (DCT 16x16), fica evidente esta ampliação na complexidade. Em relação à Tabela 4.5, o aumento no número de cálculos é superior a 269 vezes, em relação à Tabela 4.8 a ampliação no número de cálculos é superior a 44 vezes, enquanto que em relação à

Tabela 4.11 o número de cálculos é quase sete vezes maior.

Tabela 4.14 – Número de operações matemáticas da DCT 32x32 no HEVC

DCT 32x32	Multiplicações	Adições
1-D	11.008	12.928
2-D	22.016	25.856

4.2.5 Análise de Complexidade

Os resultados de número de cálculos apresentados nas seções anteriores demonstram a elevada complexidade da DCT no HEVC, deixando clara a relevância de trabalhos que visam reduzir esta complexidade.

Mas é importante salientar que a definição das transformadas no HEVC já apresentam redução enorme de complexidade se comparado à definição matemática pura, apresentada em (8) e (9). Além do uso da separabilidade, que reduz muito a complexidade da DCT, o HEVC definiu aproximações inteiras para estas transformadas, reduzindo ainda mais sua complexidade. A Tabela 4.15 apresenta um resultado comparativo do número de cálculos usados na definição matemática pura e no HEVC, onde é possível perceber as reduções significativas nos números de operações utilizadas para o cálculo da DCT pela definição do HEVC.

Tabela 4.15 – Total de operações

DCT	Versão	1-D		2-D	
		Multipl.	Somas	Multipl.	Somas
4x4	Matemática	128	120	256	240
	HEVC	32	48	64	96
8x8	Matemática	2.048	2.016	4.096	4.032
	HEVC	192	288	384	576

16x16	Matemática	32.768	32.640	65.536	65.280
	HEVC	1.408	1.856	2.816	3.712
32x32	Matemática	524.288	523.776	1.048.576	1.047.552
	HEVC	11.008	12.928	22.016	25.856

Ainda assim, é possível reduzir ainda mais estes números de cálculos usados pelo HEVC, conforme ficará claro nos próximos capítulos desta dissertação.

5 OTIMIZAÇÕES ALGORÍTMICAS NA DCT DO HEVC

A seguir, serão apresentadas as técnicas de otimizações empregadas no algoritmo das transformadas DCT do HEVC. Inicialmente, uma primeira análise foi realizada, a fim de se definir quais técnicas seriam empregadas, resultando nas heurísticas de otimização empregadas neste trabalho. Posteriormente, está descrito o algoritmo desenvolvido com a finalidade de realizar tais otimizações de forma automatizada. Também, serão apresentadas as novas equações geradas pelo software de otimização, bem como os ganhos obtidos em relação ao número de operações necessárias para se realizar os cálculos correspondentes à DCT.

5.1 Heurísticas de Otimização

Como explicado anteriormente, no HEVC o módulo das transformadas DCT possui quatro diferentes tamanhos. Como no padrão atual de codificação de vídeos, o H.264/AVC, as transformadas DCT 4x4 e DCT 8x8 já são utilizadas, decidiu-se então tomar como ponto de partida das investigações a DCT 16x16, por ser uma das duas novas propostas para o módulo das transformadas DCT, a DCT 16x16 e a DCT 32x32.

A primeira etapa deste trabalho de otimização foi identificar a transformada DCT 16x16 na versão então atual do software de referência do HEVC no momento em que se deu o desenvolvimento deste trabalho, chamado de HM4 (HEVC *Model,* versão 4.0) (JCT-VC, 2011). Então, o código fonte da DCT 1-D de 16 pontos foi localizado e investigado para gerar uma primeira versão sintetizável desta arquitetura. Ou seja, esta versão foi implementada em VHDL apenas descrevendo os cálculos envolvidos nas equações, com

operações de somas, subtrações, multiplicações e divisões, deixando a cargo da ferramenta de síntese a busca pela melhor solução arquitetural. Esta versão inicial foi gerada a partir de uma transcrição do algoritmo da DCT 1-D de 16 pontos para o VHDL sem nenhum tipo de otimização e foi realizada com o objetivo de compreender com mais detalhes o algoritmo e também para servir como base de referência para futuras comparações com as otimizações implementadas.

O HM, na versão 4.0, também foi usado como o *golden model* para o projeto apresentado neste trabalho. Deste modo, a validação arquitetural foi realizada usando dados extraídos do HM, aumentando a confiabilidade dos resultados obtidos.

Com o algoritmo da transformada 1-D completamente compreendido, passou-se por uma etapa de análise detalhada do algoritmo, o qual foi reescrito substituindo-se todas as sequências de *loops* por um novo código equivalente totalmente sequencial. Esta transformação no código foi realizada visando facilitar a futura implementação em hardware. Esta nova versão do código foi validada através de comparações dos resultados gerados com a versão original presente no código do HM.

O passo seguinte consistiu em analisar o código criteriosamente, a fim de verificar as sequências de somas/subtrações e multiplicações por constantes, de forma a encontrar partes que pudessem ser agrupadas e reutilizadas.

A fim de explicar as heurísticas de otimização desenvolvidas neste trabalho, será utilizado como exemplo a DCT de 16 pontos. Porém, as mesmas técnicas foram empregadas em todas as DCTs de diferentes tamanhos presentes no HEVC.

Os cálculos apresentados na Tabela 4.9 representam os primeiros cálculos do algoritmo correspondente à DCT de 16 pontos (etapa a_n da DCT de 16 pontos - *butterfly*). Nesta etapa, são realizadas apenas somas e subtrações sobre as entradas. Todas as demais operações e respectivas simplificações descritas a seguir foram geradas a partir dos resultados provenientes desta primeira etapa.

A equação (10), representa uma das 8 saídas específicas da DCT de 16 pontos (apresentadas na Tabela 4.10). Esta equação, assim como está

apresentada, corresponde à equação original descrita no código de referência do HEVC. Ela é composta de oito operações de multiplicações e oito operações de soma.

$$X_1 = 90*a_1 + 87*a_2 + 80*a_3 + 70*a_4 + 57*a_5 + 43*a_6 + 25*a_7 + 9*a_8 + 4) >> 3 \quad (10)$$

Uma das técnicas de otimização utilizadas neste trabalho foi a substituição das operações de multiplicação por somas e deslocamentos. Para implementações em hardware, esta técnica mostra-se extremamente vantajosa, pois os deslocamentos podem ser implementados com uma simples concatenação de zeros à direita da variável a ser multiplicada. A utilização desta técnica requer muito menos recursos de hardware do que quando utilizados multiplicadores convencionais.

A Tabela 5.1, mostra como se dá esta substituição de multiplicadores por somas e deslocamentos. Nesta tabela, constam as mesmas operações que compõem a equação (10). As colunas 'a_n' e 'Const.' correspondem à variável a ser multiplicada e à constante de multiplicação, respectivamente. Cada coluna de deslocamento corresponde a uma operação de multiplicação por uma constante de base dois, conforme indicado. E, por fim, são apresentadas as operações resultantes deste processo de otimização.

Tabela 5.1 – Substituição das multiplicações em somas e deslocamentos

a_n	Const.	Deslocamentos Utilizados (+: Somas / - : Subtrações)							Operações
		64	32	16	8	4	2	1	
		<< 6	<< 5	<< 4	<< 3	<< 2	<< 1	+/- 1	
a1	90	+		+	+		+		<< 6 + << 4 + << 3 + << 1
a2	87	+		+	+			-	<< 6 + << 4 + << 3 - 1
a3	80	+		+					<< 6 + << 4
a4	70	+			+		-		<< 6 + << 3 - << 1
a5	57		+	+	+			+	<< 5 + << 4 + << 3 + 1
a6	43		+		+			+	<< 5 + << 3 + 1
a7	25			+	+		+	+	<< 4 + << 3 + << 1 + 1
a8	9				+			+	<< 3 + 1

Como exemplo, é possível observar a linha destacada da Tabela 5.1. A multiplicação da variável a_5 pela constante 80 foi simplificada da seguinte

forma: concatenando-se 6 *bits* à direita da variável a_3, é equivalente a uma multiplicação pela constante 64 e, concatenando-se 4 *bits* à direita desta mesma variável, equivale a multiplicação pela constante 16. Somando-se o valor destas duas constantes (64 + 16), chega-se ao valor da constante de multiplicação inicial 80. Desta forma, esta operação de multiplicação foi substituída por uma operação de soma simples após deslocamentos, reduzindo enormemente o custo de implementação em hardware. Este procedimento foi realizado para todas as equações de todas as DCTs de diferentes tamanhos.

Em uma análise preliminar, pode-se deduzir que a decomposição de uma multiplicação que utilize o menor número de deslocamentos resulte em um menor número de operações e, por consequência, um menor custo de hardware. Esta conclusão está correta quando analisada uma única equação isoladamente, e é a solução mais usual (GHISSONI et al., 2011).

No entanto, neste trabalho utilizou-se, além desta otimização, o compartilhamento de subexpressões comuns entre as diferentes equações que compõem a DCT. Portanto, a otimização empregada neste trabalho utiliza não somente a substituição de multiplicações por somas e deslocamentos, mas também realiza uma análise em conjunto com esta, buscando o maior número de compartilhamentos possíveis de serem implementados. Só então, é definido quais conjuntos de deslocamentos serão utilizados. Assim, uma representação que utilize um maior número de deslocamentos poderia ser a melhor opção global caso suas equações intermediárias sejam compartilhadas em um maior número de vezes.

Esta é uma novidade apesentada neste trabalho, uma vez que alguns estudos já publicados inicialmente decompõem a multiplicação em somas e deslocamentos, buscando o menor número de deslocamentos a serem utilizados, e na sequência, usam o compartilhamento de subexpressões como um passo adicional (GHISSONI et al., 2011). Mas não foram encontrados trabalhos publicados abordando a utilização destas duas técnicas em conjunto, como empregado neste trabalho.

A implementação de forma manual das técnicas de otimização descritas acima mostra-se um trabalho árduo, mesmo para a DCT de 16 pontos, haja visto o grande número de diferentes combinações possíveis para representar as equações, considerando as diferentes representações de somas

e deslocamentos e também pela busca de compartilhamento de subexpressões. Por isso, foi desenvolvido um algoritmo para executar as referidas otimizações de forma automatizada.

A forma de implementação desta nova técnica de otimização será melhor explicada na seção 5.3 em conjunto com a descrição do algoritmo de otimização. A seguir, será exposta a complexidade da análise realizada.

5.2 Grau de Complexidade da Análise

Tratando-se da transformada DCT de 32 pontos, esta mesma análise de otimização apresentada anteriormente, sendo efetuada de forma manual, seria praticamente impossível e, caso fosse efetuada, estar-se-ia escolhendo apenas uma das milhares de soluções possíveis, não sendo possível afirmar ser esta a melhor.

Com o objetivo de apresentar o elevado número de diferentes combinações a serem avaliadas, serão tomadas como exemplo as equações específicas da DCT de 32 pontos, por esta apresentar o maior grau de complexidade e, assim, evidenciar através dos números apresentados, a importância da implementação de um algoritmo capaz de realizar estas verificações de forma automática.

Na Tabela 5.2, tem-se uma representação, dentre as diversas possíveis, da equação X_1 da DCT de 32 pontos. Da mesma forma que mostrado anteriormente, esta tabela apresenta um arranjo de somas e deslocamentos que correspondem às multiplicações da equação.

Porém, esta é apenas uma dentre as diversas formas possíveis de representar esta equação X_1. Por exemplo, a constante de multiplicação de valor 13 está representada na Tabela 5.2 pela soma de 3 diferentes deslocamentos, mas se analisada a Tabela 5.3, ter-se-iam outras 17 opções também disponíveis para representar a mesma operação de multiplicação, considerando deslocamentos de até seis *bits*. E isso ocorre da mesma forma para as demais constantes.

Tabela 5.2 - Representação da equação X1 da DCT de 32-pontos

an	Const.	Deslocamentos Utilizados (+: Somas / -: Subtrações)						
		64	32	16	8	4	2	1
		<< 6	<< 5	<< 4	<< 3	<< 2	<< 1	+1/-1
a1	90	+	0	+	+	0	+	0
a2	90	+	0	+	+	0	+	0
a3	88	+	0	+	+	0	0	0
a4	85	+	0	+	0	+	0	+
a5	82	+	0	+	0	0	+	0
a6	78	+	0	0	+	+	+	0
a7	73	+	0	0	+	0	0	+
a8	67	+	0	0	0	0	+	+
a9	61	+	0	0	-	+	0	+
a10	54	0	+	+	0	+	+	0
a11	46	0	+	0	0	+	+	0
a12	38	0	+	0	+	+	+	0
a13	31	0	+	0	0	-	+	+
a14	22	0	0	+	0	+	+	0
a15	13	0	0	0	+	+	0	+
a16	4	0	0	0	0	+	0	0

Tabela 5.3 - Diferentes representações para a constante ‘13’

an	Const.	Deslocamentos Utilizados (+: Somas / -: Subtrações)						
		64	32	16	8	4	2	1
		<< 6	<< 5	<< 4	<< 3	<< 2	<< 1	+1/-1
a15	13	0	0	0	+	+	0	+
a15	13	0	0	0	+	+	+	-
a15	13	0	0	+	-	+	0	+
a15	13	0	0	+	-	+	+	-
a15	13	0	0	+	0	-	0	+
a15	13	0	0	+	0	-	+	-
a15	13	0	0	+	0	0	-	-
a15	13	0	+	-	-	+	0	+
a15	13	0	+	-	-	+	+	-
a15	13	0	+	-	0	-	0	+
a15	13	0	+	-	0	-	+	-
a15	13	0	+	-	0	0	-	-
a15	13	+	-	-	-	+	0	+
a15	13	+	-	-	-	+	+	-
a15	13	+	-	-	0	-	0	+
a15	13	+	-	-	0	-	+	-
a15	13	+	-	-	0	0	-	-

A princípio, não é possível identificar qual o melhor arranjo a ser escolhido. A única maneira de saber qual a solução mais satisfatória é testando uma a uma.

Na Tabela 5.4, são apresentadas todas as constantes de multiplicação usadas nas 16 equações específicas da DCT de 32 pontos. A coluna “Sem Filtro” mostra o número de diferentes combinações possíveis de se representar cada uma destas constantes, com no máximo 7 deslocamentos por combinação (as 7 colunas de deslocamentos da Tabela 5.3). Já a coluna “Filtro: 4” mostra o número de combinações possíveis utilizando-se, no máximo, quatro deslocamentos por constante.

A única constante que é utilizada mais de uma vez na mesma equação é a de valor ‘90’, desta forma ela aparece duas vezes na Tabela 5.4, pois como cada multiplicação pode ser representada de uma forma diferente, este fato deve ser levado em consideração nos cálculos.

Tabela 5.4 – Combinações por constantes

Constantes	Nº de Combinações	
	Sem Filtro	Filtro: 4
90	7	3
90	7	3
88	3	3
85	13	1
82	7	3
78	7	4
73	11	3
67	9	4
61	11	4
54	11	6
46	11	6
38	12	7
31	11	5
22	13	8
13	17	8
4	5	4

Como já foi exposto, a única forma de identificar a melhor combinação a ser utilizada seria realizando testes com todas as combinações possíveis. Como cada constante possui um número de combinações diferentes e em cada uma das 16 equações estão presentes todas as constantes apresentadas na Tabela 5.4, para determinar o número de diferentes arranjos de combinações possíveis a serem testadas, basta multiplicar todos os valores da coluna “Sem Filtro” da Tabela 5.4. Efetuando estes cálculos chega-se a $1{,}8 \times 10^{15}$ arranjos

possíveis. Como são 16 equações e cada uma pode assumir um arranjo diferente, seriam $(1{,}8x10^{15})^{16}$ combinações de teste possíveis, ou $1{,}2x10^{244}$ combinações possíveis.

Como trabalhar com esse número de combinações torna-se inviável, foram aplicadas heurísticas parar reduzir este número. Foi definido como quatro o limite máximo de deslocamentos a serem utilizados para representar uma constante, pois se constatou que, com a combinação de no máximo quatro deslocamentos, é possível representar a operação de multiplicação por quaisquer das constantes definidas nas DCTs do HEVC.

Com a aplicação desta heurística, o número de combinações foi reduzido para $5x10^{9}$ arranjos possíveis por equação. Decidiu-se, também, trabalhar com o mesmo arranjo em todas as 16 equações. Desta forma, o número total de arranjos a serem avaliados foi reduzido de $1{,}2x10^{244}$ para $5x10^{9}$.

Considerando, apenas como exemplo hipotético, que a análise para cada combinação levasse em média 1n ($1x10^{-9}$) segundo, o tempo gasto para realizar toda análise seria de $3{,}8x10^{227}$ anos para as $1{,}2x10^{244}$ combinações possíveis. Por outro lado, para testar as $5x10^{9}$ combinações seriam necessários apenas 5 segundos.

Dois pontos importantes se destacam aqui: com a aplicação da heurística, foi possível reduzir o número de combinações a serem analisadas de $1{,}2x10^{244}$ para $5x10^{9}$. Mas infelizmente, desta forma, muito embora os resultados tenham causado redução expressiva no número de operações, não é possível provar que a combinação eleita pelo algoritmo resulte no menor número de cálculos possível, já que um grande número de combinações estão sendo desprezadas. Os resultados finais das otimizações realizadas serão discutidos na seção 6 deste trabalho.

5.3 Algoritmo de Otimização da DCT

Devido à grande complexidade da análise de otimização em conjunto com o elevado número de diferentes combinações possíveis a serem verificadas, foi desenvolvido um algoritmo que compara os resultados de cada combinação e retorna a melhor representação de somas e deslocamentos, que

irá gerar o maior número de compartilhamento de subexpressões entre todas as equações. Assim, a partir dos resultados gerados por este algoritmo, é possível minimizar os cálculos de todas DCTs, permitindo implementações em software e hardware com muito menor complexidade e custo.

A Tabela 5.5 apresenta de que forma o algoritmo interpreta as equações a partir de uma determinada combinação a ser avaliada. Para cada uma das 16 equações da DCT de 32 pontos, existem sete linhas, cada uma correspondendo a uma diferente multiplicação pelas constantes de base dois que podem ser substituídas por deslocamentos de *bits*. Portanto, cada combinação resulta em 112 linhas (16 equações x 7 linhas). Para cada combinação gerada, o algoritmo analisa estas 112 diferentes linhas.

Tabela 5.5 – Representação didática das equações

Equação	Representação das equações
01	64*(a1+a2+a3+a4+a5+a6+a7+a8+a9) +
	32*(a10+a11+a12+a13) +
	16*(a1+a2+a3+a4+a5+a10+a14) +
	8*(a1+a2+a3+a6+a7-a9+a11+a15) +
	4*(a4+a6+a9+a10+a11+a12-a13+a14+a15+a16) +
	2*(a1+a2+a5+a6+a8+a10+a11+a12+a13+a14)+
	a4+a7+a8+a9+a13+a15
03	64*(a1+a2+a3-a9-a10-a11-a12-a13-a14) +
	...
...	...
31	...
	-a2-a4-a8+a9-a10+a13

Internamente, o algoritmo representa estas mesmas equações descritas na Tabela 5.5 de uma forma mais eficiente para realizar os cálculos. As operações são representadas conforme a Figura 5.1, na qual todos os cálculos são armazenados em uma matriz com valores de ‘-1’, ‘0’ e ‘1’, que correspondem respectivamente a uma operação de subtração, nenhuma operação e operação de soma. Na Figura 5.1 estão representadas as sete primeiras linhas da Tabela 5.5, que compõem a equação X1.

Com base nas técnicas de otimização já descritas, foram desenvolvidos e implementados no algoritmo dois tipos de busca para analisar os diferentes arranjos de combinação: a “Maior Ocorrências” e a “Prioridade

Par", que executam respectivamente a busca por maior ocorrência de pares de somas e a busca com prioridade de ocorrência de apenas um par de soma.

A seguir será descrito como foi implementada cada uma destas buscas.

<<	Variáveis															
	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16
64	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
16	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0
8	1	1	1	0	0	1	1	0	-1	0	1	0	0	0	1	0
4	0	0	0	1	0	1	0	0	1	1	1	1	-1	1	1	1
2	1	1	0	0	1	1	0	1	0	1	1	1	1	1	0	0
1	0	0	0	1	0	0	1	1	1	0	0	0	1	0	1	0

Figura 5.1 – Representação das equações no algoritmo

5.3.1 Tipo de Busca: Maior Ocorrência

A partir de um novo arranjo gerado e representado conforme apresentado na Tabela 5.5, este tipo de busca realiza uma varredura em cada uma das 112 linhas, determinando qual o par de soma que terá o maior número de compartilhamentos, ou seja, implementando-se esta soma eleita pelo algoritmo é possível utilizar o resultado desta operação em um maior número de operações subsequentes.

A ocorrência de cada possibilidade de par de soma presente em cada linha é registrado em uma matriz, como mostrado na Figura 5.2. Por exemplo, para a primeira linha da Tabela 5.5, a soma a1+a2 seria um par possível, a1+a3 um outro par e assim para os demais.

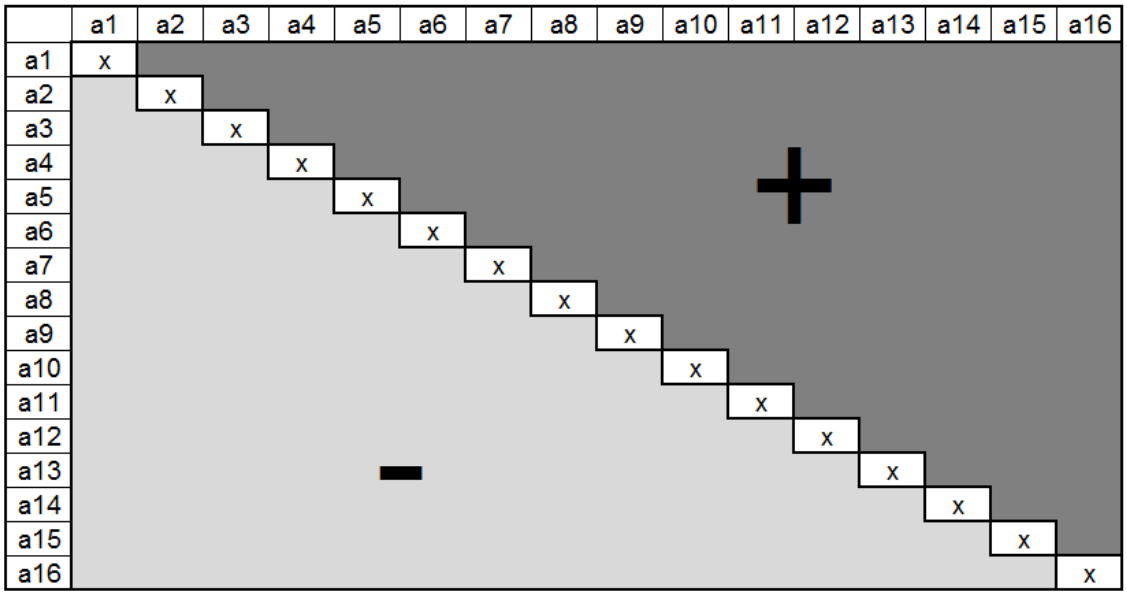

Figura 5.2 – Matriz de ocorrências do algoritmo

Na Figura 5.3 é mostrada a matriz de ocorrências parcialmente preenchida após a varredura das sete primeiras linhas, que representam a equação X_1 (Tabela 5.5).

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16
a1	x	4	3	2	3	3	2	2	1	2	2	1	1	2	1	0
a2	0	x	3	2	3	3	2	2	1	2	2	1	1	2	1	0
a3	0	0	x	2	2	2	2	1	1	1	1	0	0	1	1	0
a4	0	0	0	x	2	2	2	2	3	2	1	1	1	2	2	1
a5	0	0	0	0	x	2	1	2	1	2	1	1	1	2	0	0
a6	0	0	0	0	0	x	2	2	2	2	3	2	1	2	2	1
a7	0	0	0	0	0	0	x	2	2	0	1	0	1	0	2	0
a8	0	0	0	0	0	0	0	x	2	1	1	1	2	1	1	0
a9	1	1	1	0	0	1	1	0	x	1	1	1	1	1	2	1
a10	0	0	0	0	0	0	0	0	0	x	3	3	2	3	1	1
a11	0	0	0	0	0	0	0	0	1	0	x	3	2	2	2	1
a12	0	0	0	0	0	0	0	0	0	0	0	x	2	2	1	1
a13	0	0	0	1	0	1	0	0	1	1	1	1	x	1	1	0
a14	0	0	0	0	0	0	0	0	0	0	0	0	1	x	1	1
a15	0	0	0	0	0	0	0	0	1	0	0	0	1	0	x	1
a16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	x

Figura 5.3 – Matriz de ocorrências parcialmente preenchida

Com esta representação (Figura 5.3), é possível registrar tanto as operações de soma como as operações de subtração na mesma matriz, em

que o triângulo superior direito indica as somas e o triângulo inferior esquerdo as subtrações.

O objetivo desta busca é verificar, em todas as linhas, todas as possibilidades de somas e subtrações e ir registrando-as nesta matriz. Ao final deste processo, o algoritmo verifica qual operação teve maior ocorrência e determina que esta operação será realizada. O número de ocorrências corresponde ao número de compartilhamentos que esta operação terá.

Em um procedimento posterior, o algoritmo, após ter encontrado a operação eleita, registra esta operação em uma tabela interna conforme a Tabela 5.6. Na sequência, elimina de cada linha que possua esta operação as duas variáveis que a compõem, sendo substituídas pelo seu nome, como apresentado na Tabela 5.7.

A Tabela 5.7, apresenta o estado parcial da tabela de representação das equações, após uma primeira varredura realizada pela busca. Neste caso, a operação que apresentou a maior ocorrência foi a ‘a1+a2’. Desta forma, esta soma foi nomeada como ‘b1’ (Tabela 5.6) e todas as linhas que continham esta ocorrência foram substituídas por ‘b1’.

Tabela 5.6 – Representação das operações ‘bs’

Somas	Variáveis
b1	a1+a2
b2	...
...	...

Tabela 5.7 – Representação parcial das equações

Equação	Representação parcial das equações
01	64*(**b1** +a3+a4+a5+a6+a7+a8+a9) +
	32*(a10+a11+a12+a13) +
	16*(**b1** +a3+a4+a5+a10+a14) +
	8*(**b1** +a3+a6+a7-a9+a11+a15) +
	4*(a4+a6+a9+a10+a11+a12-a13+a14+a15+a16) +
	2*(**b1** +a5+a6+a8+a10+a11+a12+a13+a14) +
	a4+a7+a8+a9+a13+a15
03	64*(**b1** +a3-a9-a10-a11-a12-a13-a14) +
	...
...	
31	...
	-a2-a4-a8+a9-a10+a13

Finalizada esta etapa, o algoritmo zera toda a matriz de ocorrências e realiza uma nova busca, este procedimento se repete até que não reste nenhum par de variáveis ‘as’ em nenhuma linha. Então, dá-se início à nova etapa de buscas que irá gerar as somas ‘cs’ a partir das variáveis ‘bs’ juntamente com os ‘as’ remanescentes. Quando o número de ‘as’ presente em uma linha é ímpar, terá um ‘a’ que não formará nenhum par e este será repassado para a etapa seguinte para ser somado a algum ‘b’.

No pior caso, podem existir 16 variáveis ‘as’ presentes em uma mesma linha (a1,a2,a3,...,a16). E como as variáveis são agrupadas aos pares através de somas simples, como demonstrado na Tabela 5.8, é possível gerar somas ‘bs’, ‘cs’, ‘ds’ e ‘es’.

Tabela 5.8 – Etapas de busca

Busca	Níveis de Profundidade das Buscas
‘bs’	(a1+a2)+(a3+a4)+(a5+a6)+(a7+a8)+(a9+a10)+(a11+a12)+(a13+a14)+(a15+a16)
‘cs’	(b1+b2)+(b3+b4)+(b5+b6)+(b7+b8)
‘ds’	(c1+c2)+(c3+c4)
‘es’	(d1+d2)

5.3.2 Tipo de Busca: Prioridade Par

Este tipo de busca é uma derivação da busca por maior ocorrência; ela inicialmente realiza uma varredura em todas as linhas em busca da ocorrência de apenas um par de variáveis presente em alguma linha. Caso encontre, como esta operação é a única possível nesta linha, ela terá de ser efetuada necessariamente, mesmo que não haja nenhuma outra ocorrência em nenhuma outra linha. Desta forma, caso o algoritmo encontre a ocorrência de apenas um par de soma, ele determina esta operação como a eleita e faz, então, uma nova varredura em todas as linhas, buscando a ocorrência desta operação.

Os demais procedimentos são similares aos da busca por maior ocorrência. Caso o algoritmo não encontre em nenhuma linha apenas um par de variáveis, então é executada a busca por maior ocorrência.

5.3.3 Comparação Entre os Tipos de Busca

Foram realizados testes iniciais para identificar qual dos dois tipos de busca apresentaria os melhores resultados. Para tanto, foi implementada uma versão parcial do algoritmo executando os dois tipos de buscas apenas para a etapa de buscas ‘bs’. Para esta verificação inicial, foi processada a análise de 10.000.000 arranjos de diferentes combinações. Os dados obtidos são apresentados na Figura 5.4.

Neste gráfico (Figura 5.4), o ‘eixo y’ apresenta o total de códigos que geraram um determinado número de operações ‘bs’ e o ‘eixo x’ mostra a variação do número total de operações ‘bs’ para as buscas realizadas. Analisando este gráfico, percebe-se que a curva da busca por prioridade par (P. Par) mostra-se mais à esquerda do que a curva da busca por maior ocorrência (Maior O.) Isto mostra que, na maioria dos casos, a busca por prioridade par se mostrou mais eficiente, utilizando um número menor de somas.

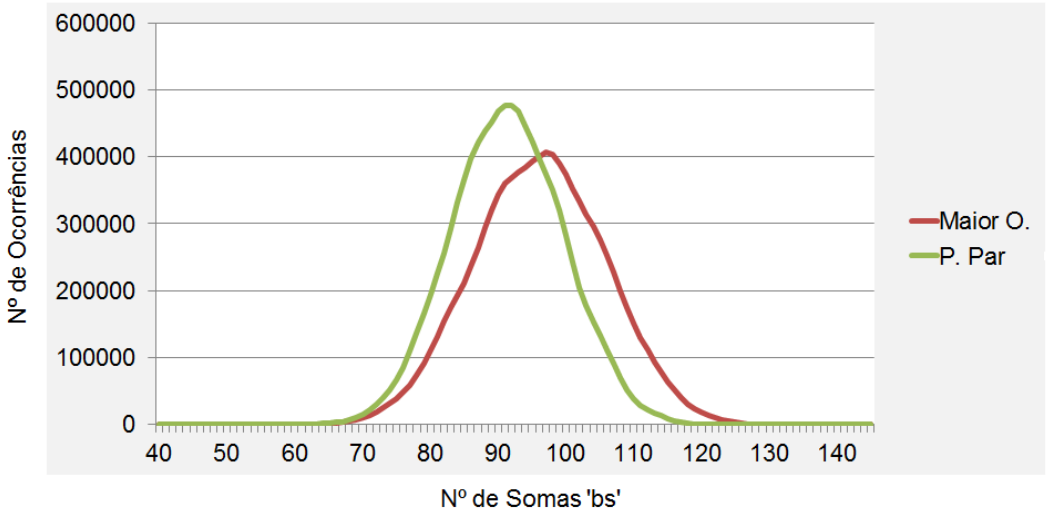

Figura 5.4 – Gráfico comparativo entre os dois tipos de busca

Na Figura 5.4, é possível perceber que o número máximo de ocorrências para a busca por prioridade par deu-se com valores em torno de 93 somas ‘bs’; já para a busca por maior ocorrência, este valor foi de 99 somas ‘bs’. Para cada código verificado, foi analisado qual tipo de busca obteve o melhor resultado, ou seja, resultou em um menor número de somas.

Como mostrado na Figura 5.5, a busca por prioridade par mostrou-se muito mais eficaz que a busca por maior ocorrência. Se somada a porcentagem de ganho com os valores de empate, a busca por prioridade par mostra-se melhor ou igual em 88% das buscas. Com base nestes dados, inicialmente poderia se tomar a decisão de utilizar apenas a busca por prioridade par, por se mostrar mais eficiente. E desta forma, executando apenas um tipo de busca por código, o tempo computacional diminuiria muito.

Mas, realizando uma análise mais detalhada nos dados e verificando apenas os melhores resultados das buscas, como apresentado na Figura 5.6, é possível constatar que, embora a busca por prioridade par apresente o melhor resultado em 88% das buscas, o melhor resultado final para as 10.000.000 verificações foi obtido pela busca por maior ocorrência, resultando em 41 somas ‘bs’ contra 45 somas ‘bs’ obtidas pela busca por prioridade par.

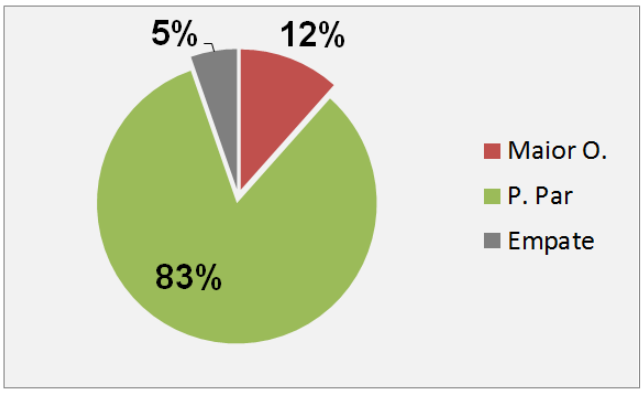

Figura 5.5 – Percentual de ganho entre os dois tipos de busca

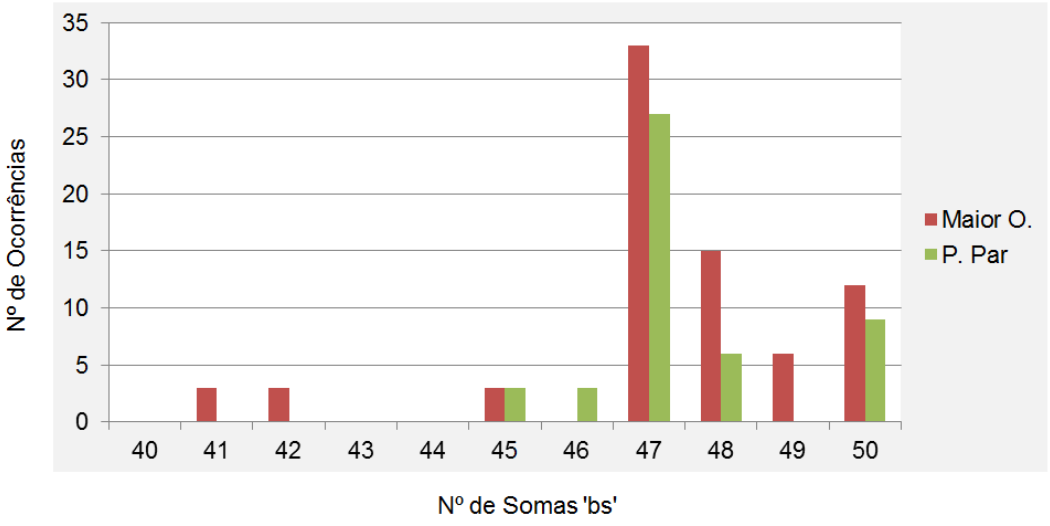

Figura 5.6 – Comparativo mais detalhado entre os dois tipos de buscas

Como a intenção do algoritmo é detectar o menor número de somadores possíveis de representar as operações avaliadas e como a partir dos testes realizados não foi possível afirmar qual tipo de busca efetivamente apresentaria o melhor resultado final, decidiu-se então manter os dois tipos de busca no algoritmo desenvolvido.

5.3.4 Propostas Algorítmicas Iniciais

Implementou-se, inicialmente, quatro propostas de versões algorítmicas, conforme explicado a seguir:

Proposta 01: Nesta proposta, o algoritmo exerce a busca completa, realizando os dois tipos de busca para todas as etapas (‘bs’, ‘cs’, ‘ds’ e ‘es’). Dentre as quatro propostas de busca, esta garante o melhor resultado, não desprezando nenhuma das possibilidades, porém também é a que apresenta o maior custo computacional. Devido ao grande número de códigos a serem verificados, esta proposta acaba não se mostrando a melhor solução, pois para cada arranjo de combinação acaba tendo que percorrer 16 percursos diferenciados.

Proposta 02: Esta configuração é muito parecida com a proposta 01, porém após cada etapa de busca, foram inseridos filtros, com valores pré-determinados, de modo a permitir a escolha do número máximo de operações permitidas em cada etapa de busca, descartando os piores resultados já no meio do processo. Quanto maior o valor definido para cada filtro, mais similar ficará esta proposta se comparada com a proposta 01 e quanto menores os valores dos filtros, mais códigos serão eliminados das buscas subsequentes, aumentando a velocidade da busca, porém eliminando possíveis candidatos a melhor resultado.

Proposta 03: esta proposta realiza os dois tipos de busca em cada etapa, depois compara os resultados e passa para a etapa seguinte apenas o resultado vencedor.

Proposta 04: nesta proposta, além de ser executado o mesmo procedimento que a proposta 03, ainda foram inseridos filtros após a escolha do melhor resultado de cada busca. Assim como na proposta 02, o valor

definido para os filtros é determinante tanto para o resultado final como para o desempenho do algoritmo.

5.3.5 Versão Final do Algoritmo de Otimização

Após uma série de testes, foi possível verificar que nem sempre um menor resultado em uma etapa de busca inicial resulta no melhor resultado final. Desta forma, como as propostas 03 e 04 desprezam um grande número de combinações, as mesmas foram descartadas. E entre a proposta 01 e a proposta 02, a segunda mostrou-se mais versátil, já que a depender do ajuste dos filtros, pode-se ter desde a busca completa até uma grande velocidade de processamento.

Inicialmente, começou-se a trabalhar com esta versão, a proposta 02. Mas, para a última etapa de busca, a de somas ‘es’, foi possível constatar que seria desnecessário manter os dois tipos de busca, pois em todas as combinações que chegassem até este estágio, a única possibilidade de ocorrência seria de apenas uma soma possível. Portanto, independente do tipo de busca, o resultado será exatamente o mesmo.

Através de testes complementares, constatou-se que a etapa de busca ‘ds’ também apresentou resultados muito semelhantes para ambos os tipos de buscas. Isso se deve ao número reduzido de possibilidades de operações que as etapas finais apresentam, ficando os casos críticos concentrados para as etapas de buscas ‘bs’ e ‘cs’.

Através destas análises, chegou-se à versão final do algoritmo que foi implementado, o qual mantém apenas um tipo de busca para as etapas ‘ds’ e ‘es’. A busca escolhida para estas etapas finais foi a prioridade par, por apresentar um menor custo computacional. Esta versão do algoritmo está apresentada na Figura 5.7.

Nas Figura 5.7, a busca por maior ocorrência e a busca por prioridade par, são identificadas respectivamente como ‘busca 0’ e ‘busca 1’.

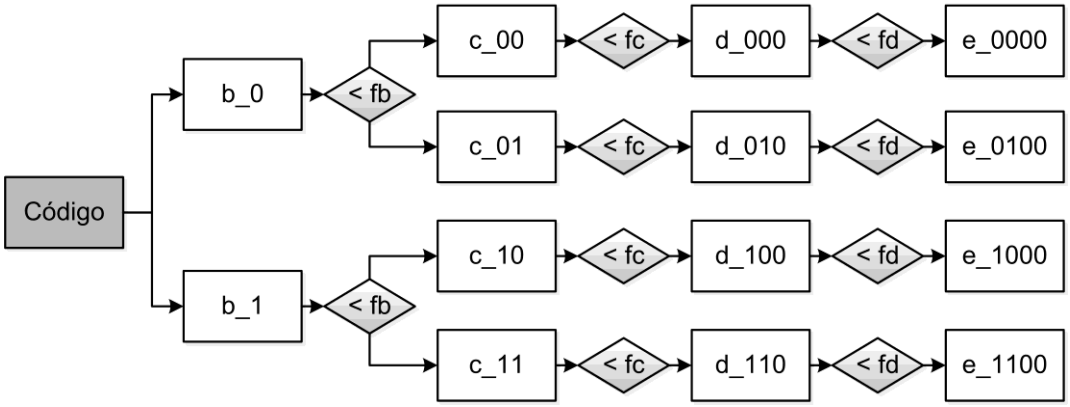

Figura 5.7 – Fluxograma do algoritmo de busca

Com esta nova configuração do algoritmo, é possível realizar a verificação de cada código com um total de, no máximo, 14 buscas, ao contrário da proposta 02, que realizava a mesma verificação com um total de, no máximo, 30 buscas por código. Esta nova configuração apresenta bons ganhos em termos de custo computacional e com praticamente nenhuma perda de qualidade do resultado. Não foi possível gerar dados precisos sobre a perda de qualidade gerada, já que a versão completa do algoritmo, sem heurísticas para redução de complexidade, não é possível de ser implementada. Ainda assim, pelos testes realizados, é possível inferir que a perda, se existir, será muito pequena.

Além dos filtros de buscas, o algoritmo possui outros filtros, destinados ao armazenamento dos melhores resultados. Em cada etapa de busca, o algoritmo armazena os dados dos códigos que apresentam resultados inferiores aos valores definidos nos filtros de armazenamento.

Como o objetivo do algoritmo é obter o menor hardware, foi implementada uma última verificação, para todos os códigos que chegam até a etapa final de busca. Para estes casos, é realizada uma análise quantitativa total de todas as etapas e, considerando que em uma implementação em hardware a cada etapa subsequente será utilizado um somador com um número maior de *bits*, este dado também é considerado pelo algoritmo.

Então, basicamente, o cálculo realizado é uma multiplicação do número de somas encontradas em cada etapa pelo número de *bits* utilizados nos somadores destas etapas e uma posterior soma de todos os dados. Este valor final acaba por representar parcialmente o tamanho do hardware a ser

implementado, então é à busca por este menor valor final que o algoritmo se destina.

Concluída a definição do algoritmo, foi gerado um software todo escrito em linguagem C com a plataforma NetBeans (NETBEANS, 2011) e processado no sistema operacional Linux.

O algoritmo foi devidamente validado, através da implementação de um módulo auxiliar no próprio software, no qual, após terem sido definidas todas as operações criadas pelo software é, então, chamada uma função que compara os resultados destas saídas com as geradas pelas equações do software de referência do HEVC.

Tabela 5.9 – Dados gerados pelo GProf

Tempos (segundos)			Chamadas	Função
%	Acumulado	Por Função		
67.30	6.19	6.19	19678	maior_ocorrencia
23.59	8.36	2.17	2203936	verifica_deslocamento
7.39	9.04	0.68	28698	verifica_existencia_par
0.65	9.10	0.06	300	conf_inicial
0.43	9.14	0.04	300	conf_final
0.33	9.17	0.03	28398	seta_variaveis
0.33	9.20	0.03	8720	prioridade_par
0.00	9.20	0.00	300	busca
0.00	9.20	0.00	160	arquiva
0.00	9.20	0.00	160	validacao
0.00	9.20	0.00	160	vhdl
0.00	9.20	0.00	10	modulo3
0.00	9.20	0.00	10	modulo4
0.00	9.20	0.00	1	m1_geraCombi
0.00	9.20	0.00	1	m2_geraCombi
0.00	9.20	0.00	1	modulo1
0.00	9.20	0.00	1	modulo2

Após a validação, constatou-se que, embora o software estivesse realizando todas as buscas de forma correta, o tempo computacional ainda estava muito elevado. Foi, então, utilizada a ferramenta GProf (GNU, 2011), disponível gratuitamente no sistema operacional Linux (Ubuntu versão 12.04 LTS), que se destina a analisar o número de chamadas e o tempo gasto por cada função de um determinado software. Depois de submetido a este tipo de

análise, os dados apresentados pelo software de otimização estão apresentados na Tabela 5.9.

Com os dados gerados pela ferramenta GProf, foi possível constatar que 98,28% do custo computacional de todo o software está concentrado em apenas três funções (destacadas na Tabela 5.9). Desta forma, foi possível concentrar a análise e otimizações do software nestas funções mais críticas.

Na Figura 5.8, é mostrado um gráfico de chamadas das principais funções que compõem o software. As funções ‘main’, ‘modulo1’, ‘m1_geraCombi’ e ‘modulo2’ são executadas apenas uma única vez durante todo o processo de busca e são responsáveis por configurações iniciais.

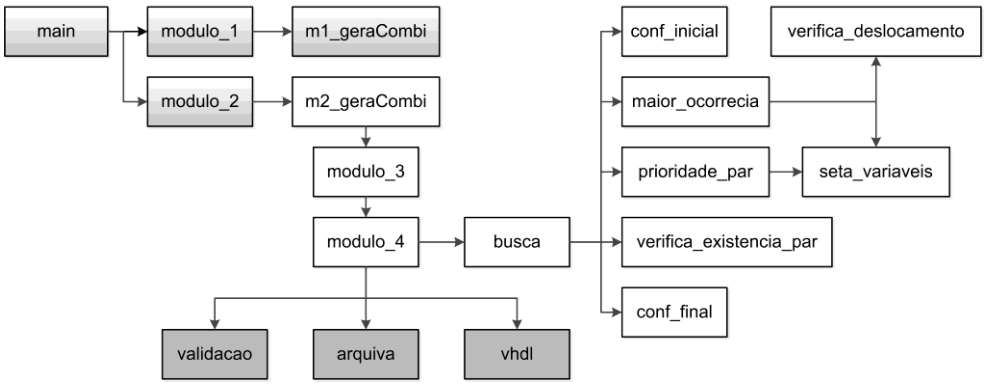

Figura 5.8 – Gráfico de chamadas das funções

As funções ‘validacao’ e ‘vhdl’ são funções auxiliares e não são utilizadas durante todo o processo de busca. A função ‘arquiva’ só é chamada caso a busca encontre um código que resulte um número menor que os definidos nos filtros. As demais funções são executadas em todo o processo de busca.

A função ‘m2_geraCombi’ é responsável por gerar as diferentes combinações a serem testadas; o ‘modulo3’ organiza os dados na forma que foi apresentado na Tabela 5.5; o ‘modulo4’ é responsável por chamar a função ‘busca’ já verificando os valores dos filtros e informando qual a etapa de busca que será realizada.

A função ‘busca’ é a responsável por toda a pesquisa propriamente dita, ela foi desenvolvida de forma que esta mesma função seja capaz de realizar as buscas independente de que estágio esteja sendo realizada a

verificação. Através das informações passadas pelo ‘modulo4’, a função ‘busca’ é automaticamente reconfigurada para realizar a busca desejada. As demais funções chamadas pela função ‘busca’ são funções internas desta e são chamadas conforme a necessidade.

Outra funcionalidade implementada no software foi a realização de buscas de forma paralela, ou seja, foi possível dividir o número total de arranjos de combinações e executar as buscas em diversos computadores simultaneamente, bem como vários processos em paralelo em um mesmo computador

Através da relação dos computadores disponíveis no grupo de pesquisa GACI e as correspondentes configurações (Tabela 5.10), definiu-se como sendo possível, devido à disponibilidade, a execução de até 60 processos em paralelo.

Com base nestes dados, foram realizados mais alguns testes com a finalidade de se determinar os valores dos filtros a serem adotados nas buscas. O ajuste destes filtros é extremamente delicado e influencia diretamente tanto no resultado final como no tempo gasto no processamento.

Tabela 5.10 – Computadores disponíveis

ID	Processador	Processos em Paralelo	Memória
1	Core 2 Quad Q6600	2	2Gb
2	Core 2 Quad Q9650	2	4Gb
3	Core 2 Quad Q9650	2	4Gb
4	Core 2 Quad Q9650	2	4Gb
5	Core I5 760	4	4Gb
6	Core I5 760	4	4Gb
7	Core I5 760	4	4Gb
8	Core I5 750	4	4Gb
9	Core I7-2600	8	8Gb
10	Core I5 760	4	4Gb
11	Core I5 760	4	4Gb
12	Core I5 760	4	4Gb
13	Core 2 Quad Q6600	2	2Gb
14	Core i5-2400	4	8Gb
15	Core i5-2400	4	8Gb
16	Core i5-2400	4	8Gb
17	Core i5-2400	4	8Gb
18	2x Xeon E5600 Hexa	24	24Gb

Na Tabela 5.11 são apresentados os resultados destes testes, que foram gerados executando-se o software de otimização para 10.000 arranjos de combinações. Na última coluna da tabela é apresentado a estimativa de dias necessários para realizar as buscas em todos os $5x10^9$ códigos, levando-se em consideração a execução de 60 buscas em paralelo.

Tabela 5.11 – Valores dos Filtros x Tempos Estimados

Filtros		Tempo Médio p/ cód. (seg)	Tempo Total (dias)	Tempo Total c/ 60 proc. em paralelo (dias)
bs	cs			
55	120	0,0037	214,82	3,58
60	120	0,0057	330,95	5,52
70	125	0,0057	330,95	5,52
80	135	0,0169	981,23	16,35
80	**150**	**0,0269**	**1561,84**	**26,03**
90	140	0,05	2903,04	48,38
100	150	0,119	6909,24	115,15
nulo*	nulo*	0,24	13934,59	232,24
Completa**		0,77	44706,82	745,11

* Sem aplicação de filtros
** Busca Completa (Proposta 01)

Com base nos dados da Tabela 5.11 e levando-se em consideração o tempo de execução do software e a qualidade final dos resultados, foram definidos os valores dos filtros “bs” e “cs” como 80 e 150, respectivamente, e foi possível realizar a verificação de todos os arranjos de combinações em torno de 26 dias.

O tempo gasto para cada código verificado é, em média, menor que 0,027 segundos. Considerando o grande número de verificações internas efetuadas em cada código, este tempo não é desprezível.

Uma vez concluídas as buscas para a DCT de 32 pontos, que é a de maior complexidade, a alteração do software para analisar as demais DCTs deu-se de forma extremamente simples, pois durante todo o desenvolvimento já se buscou esta facilidade de adaptação ao tamanho da transformada a ser analisada.

Para a DCT de 16 pontos, o tempo de processamento deu-se em aproximadamente 12 horas, com apenas quatro processos em paralelo. Para a

DCT de 8 pontos e para a DCT de 4 pontos, o tempo gasto foi menor que um minuto e menor que um segundo, respectivamente, com apenas um processo de busca.

O alto custo computacional do software de otimização acaba sendo recompensado pelos ganhos expressivos relacionados à redução de complexidade das transformadas tanto para implementações em software quanto, principalmente para implementações em hardware. Estes ganhos serão discutidos na próxima seção.

5.4 Equações Geradas pelo Software

Para as equações otimizadas pelo software, são considerados como dados de entrada os resultados das operações de subtração do bloco *butterfy*.

O software de otimização, após analisar as diferentes combinações, apresentou como resultado as operações descritas no Apêndice A e brevemente explicadas a seguir, tomando como exemplo a DCT de 4 pontos.

Para a DCT de 4 pontos, das 4 equações que compõem esta DCT (Tabela 4.4), apenas 2 delas são otimizadas pelo software. As outras duas, por apresentarem uma mesma constante de multiplicação cujo valor é uma potência de base 2 e portanto passível de ser substituída em hardware por um simples deslocamento, foram simplificadas de forma diferente e serão apresentadas posteriormente na descrição das arquiteturas em hardware.

Desta forma, a análise do software segue um mesmo padrão para todas as DCTs de diferentes tamanhos, sendo considerados como dados de entrada apenas os resultados das operações de subtração do bloco *butterfly* da DCT correspondente.

A Tabela 5.12 a Tabela 5.13 apresentam os cálculos gerados pelo software de otimização para a DCT de 4 pontos.

Tabela 5.12 – Segunda etapa de operações da DCT de 4 pontos (bs)

Etapa 'bs' da DCT de 4 pontos	
b1 = a1 + a2	b2 = a1 - a2

Tabela 5.13 – Operações finais geradas pelo software da DCT de 4 pontos

EOS da DCT de 4 pontos		
Saída_2	=	64*b1 + 32*b2 - 16 a1 + 4*b1 - a1
Saida_4	=	64*b2 - 32*b1 + 16*a2 + 4*b2 + a2

O total de operações utilizadas para os cálculos da DCT de 4 pontos está apresentado na Tabela 5.14.

Tabela 5.14 – Total de operações geradas pelo software da DCT de 4-pontos

DCT 4-pontos	Multiplicações	Adições
1D	0	80
2D	0	160

As equações geradas pelo software de otimização para as demais DCTs, bem como o total de operações utilizadas em cada uma, estão descritos em sua totalidade no Apêndice A.

5.5 Análise dos Ganhos Gerados pelas Otimizações

Reunindo o número total de operações necessárias para realizar os cálculos das DCTs de diferentes tamanhos, tanto das versões originais descritas na definição matemática e no HEVC, como também das versões otimizadas, chega-se aos dados apresentados na Tabela 5.15. É importante destacar que a Tabela 5.15 apresenta o número de operações para as transformadas 1-D e 2-D.

O primeiro resultado importante da Tabela 5.15 é a expressiva redução de complexidade das transformadas definidas pelo HEVC se comparadas à definição matemática. Estes ganhos expressivos são função, principalmente, do uso da propriedade da separabilidade.

Tabela 5.15 – Total de operações

DCT	Versão	1D		2D	
		Mult.	Somas	Mult.	Somas
4x4	Matemática	128	120	256	240
	HEVC	32	48	64	96
	Otimizado	0	80	0	160
8x8	Matemática	2.048	2.016	4.096	4.032
	HEVC	192	288	384	576
	Otimizado	0	512	0	1.024
16x16	Matemática	32.768	32.640	65.536	65.280
	HEVC	1.408	1.856	2.816	3.712
	Otimizado	0	3.008	0	6.016
32x32	Matemática	524.288	523.776	1.048.576	1.047.552
	HEVC	11.008	12.928	22.016	25.856
	Otimizado	0	19.680	0	39.360

Como se sabe, o custo tanto de software como de hardware de uma operação de multiplicação é muito maior que uma operação de soma. Desta forma, pode-se constatar pelos dados apresentados na Tabela 5.15, que os ganhos obtidos em redução do custo operacional apresentados pelas versões otimizadas mostram-se expressivos quando comparados com as versões descritas no HEVC, revelando que as soluções resultantes das otimizações apresentam grandes reduções nas DCTs de diferentes tamanhos, tanto para as DCTs 1-D como para as DCTs 2-D.

Para ficarem mais claros os impactos da transformação dos multiplicadores em deslocamentos e somas, os números de operações do HEVC e da versão otimizada apresentados na Tabela 5.15 foram convertidos em números de *bits* de soma (somadores completos) necessários para se realizar tais operações. Para tanto, considerou-se como sete o número de *bits* necessários para representar cada constante de multiplicação. E cada operação de multiplicação foi convertida para somas equivalentes para representar seus cálculos.

Esta conversão das multiplicações, deu-se da seguinte forma: para uma multiplicação de (m x n) *bits*, o número de *bits* de somas equivalentes é igual a (m x (n-1)) *bits*. E o resultado final possui um tamanho de (m + (n-1)) somadores completos.

Para ambas soluções, tanto nas versões originais do HEVC como nas versões otimizadas, levou-se em consideração o menor número possível de somadores completos por estágio e o acréscimo de 1 *bit* após cada estagio de soma, para evitar o *overflow*. Também foi considerado, por simplificação, que o custo dos somadores é igual ao custo dos subtratores.

O número total de somadores completos para se representar todas as operações de ambas versões em todas as DCTs está apresentado na Tabela 5.16, considerando o número de somadores completos usados na primeira DCT 1-D, na segunda DCT 1-D e na DCT 2-D.

Tabela 5.16 – Número total de somadores completos utilizados

DCT	Versão	Somadores de um *bit* utilizados			
		1ª 1-D	2ª 1-D	2-D	Ganho%
4x4	HEVC	2.464	4.912	7.376	-
	Otimizada	1.032	1.752	2.784	62,26
8x8	HEVC	15.360	30.048	45.408	-
	Otimizada	6.800	11.408	18.208	59,90
16x16	HEVC	110.912	216.320	327.232	-
	Otimizada	40.992	68.064	109.056	66,67
32x32	HEVC	851.968	1.661.824	2.513.792	-
	Otimizada	270.816	452.544	723.360	71,22

Através dos resultados apresentados na Tabela 5.16 é possível perceber que os ganhos em redução de somadores completos são superiores a 59% para as DCTs 2-D de todos os tamanhos. O uso das operações geradas pelo software de otimização permite uma redução entre 2,5 vezes a 3,5 vezes no número de somadores completos em relação à abordagem do HEVC. Isto significa que a utilização das otimizações propostas permite uma redução expressiva na complexidade do módulo das transformadas no HEVC, contribuindo para reduzir a complexidade do codificador e do decodificador de vídeo do padrão emergente de codificação de vídeos HEVC.

6 ARQUITETURAS PROJETADAS PARA AS DCTS DO HEVC

Foram implementas as arquiteturas das DCTs de todos os tamanhos que compõem o módulo das transformadas no padrão emergente de codificação de vídeos HEVC. Além das transformadas DCT 1-D de tamanho 4, 8, 16 e 32 pontos, também foi implementada uma arquitetura chamada de "Multi-DCT", que é capaz de realizar todos os cálculos de qualquer uma das DCTs de diferentes tamanhos, conforme Figura 6.7.

Após da construção das transformadas 1-D, também foram implementadas as transformadas 2-D, seguindo a configuração mostrada na Figura 4.5, para todos os tamanhos definidos no HEVC: 4x4, 8x8, 16x16 e 32x32, além da transformada 2-D Multi-DCT.

A seguir serão apresentadas as arquiteturas implementadas das DCTs de diferentes tamanhos, assim como da Multi-DCT.

6.1 Descrição das Arquiteturas

Nesta seção serão descritas as arquiteturas implementadas das transformadas DCT propostas pelo HEVC, com base nas otimizações algorítmicas geradas pelo software de otimização desenvolvido para esta finalidade.

As descrições em VHDL das transformadas DCT 1-D de diferentes tamanhos foram geradas automaticamente através da função 'vhdl' do software de otimização. Uma vez escolhida a melhor combinação, a função 'vhdl' é

então chamada e gera um arquivo ASCII' com toda a descrição da arquitetura a ser implementada na linguagem VHDL. Esta descrição automática das DCTs 1-D é gerada de forma completamente combinacional. Já para a versão completa da DCT 2-D, foi replicado o código da DCT 1-D alterando-se apenas o tamanho de *bits* de cada sinal, os valores utilizados na etapa final de arredondamento e inserindo-se uma matriz intermediária de transposição.

Todas as arquiteturas foram descritas em VHDL e sintetizadas em FPGA da Altera utilizando a ferramenta de síntese Quartus II. O FPGA escolhido para as implementações foi o Stratix V, por apresentar desempenhos elevados. O modelo do dispositivo FPGA escolhido foi o 5SGXMABN3F45I4.

Para todas arquiteturas implementadas, as DCTs de diferentes tamanhos e a Multi-DCT, também foi gerada uma arquitetura de referência, sem otimizações, para avaliar melhor os impactos das otimizações realizadas nos algoritmos das DCTs. Esta arquitetura de referência, chamada de "original" neste texto, foi implementa diretamente em VHDL a partir das equações definidas no HM 9 (JCT-VC , 2012) em uma descrição puramente comportamental.

6.1.1 DCT 4x4

A arquitetura da DCT 4x4 apresenta a mesma configuração que foi apresentada na Figura 4.6. Ela é composta de 4 entradas, as operações que fazem parte da primeira DCT 1-D de 4 pontos, uma matriz intermediária de transposição 4x4 e as operações da segunda DCT 1-D, assim completando todos os cálculos necessários para gerar a DCT 2-D.

A Figura 6.1 mostra a arquitetura da DCT 1-D de 4 pontos. As 4 entradas, de 9 *bits*, são submetidas à primeira etapa de operações, geradas pelo *butterfly* e os resultados das operações de subtração, com 10 *bits*, são passados como entrada para as Equações Otimizadas pelo Software – EOS. As operações internas ao bloco EOS podem ser consultadas no Apêndice A.

O resultado das operações de soma do bloco *butterfy,* também com 10 *bits,* são somados e subtraídos e, então são concatenados 6 *bits* à direita dos resultados, equivalendo a uma multiplicação pela constante de valor 64.

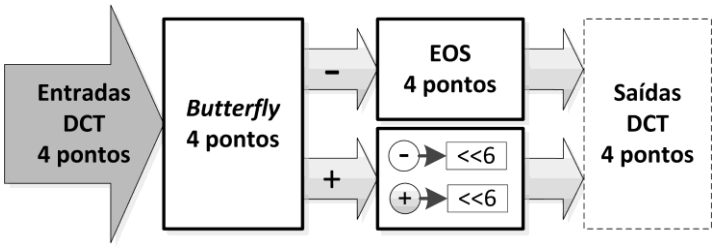

Figura 6.1 – Arquitetura 1-D da DCT 4x4

Todos os resultados, ou seja, saídas 1, 2, 3 e 4, passam, então, pelo procedimento final de arredondamento, que não está apresentado na Figura 6.1. Todas saídas possuem 16 *bits*. Para estas arquiteturas, esta etapa final de arredondamento foi implementada descrevendo-se os cálculos diretamente como no exemplo da equação (11), pois foi constatado que a ferramenta de síntese (Quartus II) é capaz de realizar uma otimização interna mais vantajosa desta forma.

$$X_1 = (Demais\ operações\ + add)\ /\ shift \tag{11}$$

Como a segunda DCT 1-D executa os mesmos cálculos que a primeira DCT 1-D, para completar a arquitetura da DCT 4x4, ainda resta apresentar a matriz de transposição. A implementação desta matriz também sofreu uma análise e um processo de otimização.

A Figura 6.2 apresenta a arquitetura da matriz de transposição. E1, E2, E3 e E4 são as entradas da matriz e correspondem às saídas da 1ª DCT de 4 pontos. S1, S2, S3 e S4 representam as saídas da matriz de transposição e estes dados são fornecidos como entrada para a 2ª DCT 1-D. A cada 4 ciclos de *clock,* o sinal de controle dos multiplexadores é alterado e os dados de leitura e escrita são comutados de linha para coluna e vice-versa. Desta forma, mesmo utilizando-se apenas uma única matriz de registradores, é possível realizar, simultaneamente, as operações de leitura e escrita gerando a transposição característica da DCT.

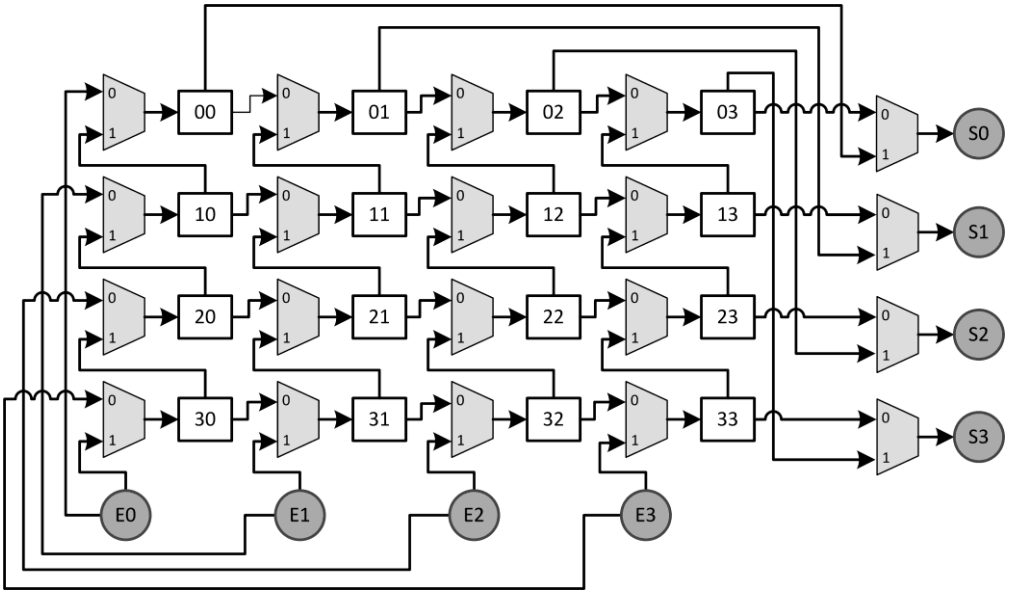

Figura 6.2 – Matriz de transposição da DCT 4x4

Na Figura 6.3, é apresentada a arquitetura completa da DCT 4x4, composta pelas duas instâncias da DCT 1-D, e a matriz de transposição.

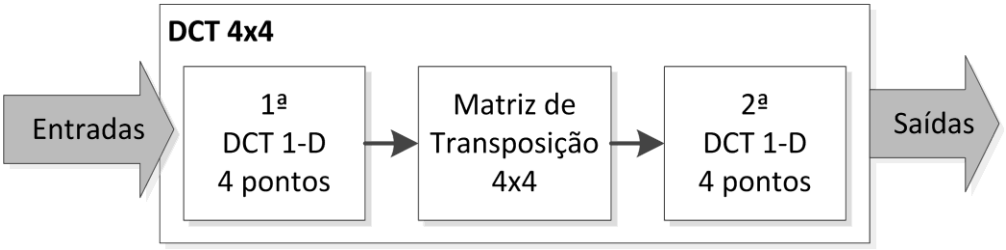

Figura 6.3 – Arquitetura completa da DCT 4x4

As arquiteturas parciais, tanto da 1ª DCT 1-D como da 2ª DCT 1-D, são totalmente combinacionais. Esta decisão foi tomada para simplificar o projeto em hardware da primeira versão arquitetural. Além disso, este tipo de implementação permite uma latência reduzida nestes cálculos. A arquitetura da transformada 2-D utiliza registradores na matriz de transposição. A arquitetura da transformada 4x4 utiliza o máximo paralelismo possível, consumindo quatro amostras por ciclo de clock.

Os dados de síntese da arquitetura da DCT 4x4 estão expressos na Tabela 6.1, bem como a arquitetura intermediária 1ª DCT 1-D.

Tabela 6.1 – Dados de Síntese da DCT 4x4

DCT 4x4	Versão Otimizada			
	1ª DCT 1-D	Matriz de Transposição	2ª DCT 1-D	DCT 2-D
ALMs	128	151	210	471
Registradores	0	269	0	283
Frequência	128,92 MHz	580.05 MHz	114,05 MHz	108,25 MHz

(Stratix V - 5SGXMABN3F45I4) – (Sem restrições de síntese)

6.1.2 DCT 8x8

A descrição da DCT 8x8, assim como os demais tamanhos, seguem o mesmo padrão descrito na DCT 4x4. Portanto serão explicados de forma mais sucinta.

A arquitetura da DCT 1-D de 8 pontos está apresentada na Figura 6.4.

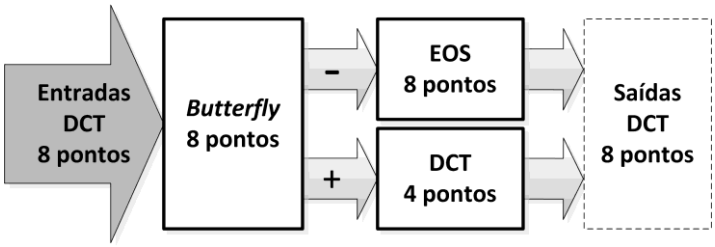

Figura 6.4 – Arquitetura 1-D da DCT 8x8

A estrutura da DCT de 8 pontos consiste de 8 entradas que passam, inicialmente, pelo bloco *butterfly*. O resultado das operações de somas do bloco *butterfly* é repassado como entrada para o bloco DCT 4 pontos, que possui internamente toda a arquitetura já apresentada no item anterior. O resultado das operações de subtração do bloco *butterfly*, é passado como entrada para o bloco EOS, que apresenta internamente as operações geradas pelo software de otimização que correspondem às 4 equações específicas da DCT de 8 pontos (Apêndice A).

Os dados de síntese da arquitetura DCT 8x8 estão apresentados na Tabela 6.2.

A arquitetura da DCT 8x8 também foi desenvolvida com as duas instâncias de DCT 1-D projetadas de forma puramente combinacional, conectadas com a matriz de transposição. A arquitetura da DCT 8x8 consome

oito amostras por ciclo de *clock*, que é o máximo paralelismo possível quando a separabilidade é usada em uma transformada 8x8..

Tabela 6.2 – Dados de síntese da DCT de 8 pontos

DCT 8x8	Versão Otimizada			
	1ª DCT 1-D	Matriz de Transposição	2ª DCT 1-D	DCT 2-D
ALMs	454	573	672	1.622
Registradores	0	1.093	0	1.130
Frequência	93,65 MHz	580.05 MHz	80,72 MHz	78,97 MHz

(Stratix V - 5SGXMABN3F45I4) – (Sem restrições de síntese)

6.1.3 DCT 16x16

A arquitetura da DCT de 16 pontos está representada na Figura 6.5. Esta arquitetura foi desenvolvida de forma puramente combinacional e foi replicada para gerar a DCT 16x16. A conexão das duas instâncias de DCT 1-D é feita através de uma matriz de transposição de tamanho 16x16, composta por blocos de registradores.

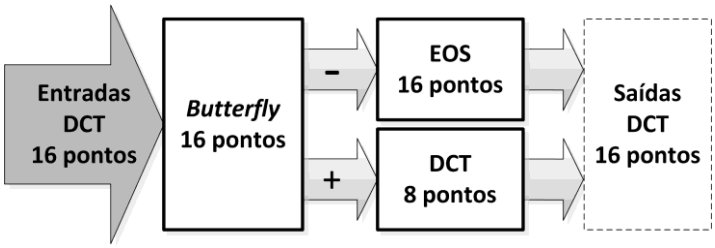

Figura 6.5 – Arquitetura 1-D da DCT 16x16

A arquitetura da DCT de 16 pontos reusa a arquitetura da DCT de 8 pontos apresentada na seção anterior e consome 16 amostras por ciclo de *clock*.

Os dados de síntese, da arquitetura descrita em VHDL com as operação otimizadas (Apêndice A), estão descritos na Tabela 6.3.

Tabela 6.3 – Dados de síntese da DCT 16x16

DCT 16x16	Versão Otimizada			
	1ª DCT 1-D	Matriz de Transposição	2ª DCT 1-D	DCT 2-D
ALMs	1.304	2.175	1.903	5.256
Registradores	0	4.413	0	4.437
Frequência	79,11 MHz	423,37 MHz	71,43 MHz	69,80 MHz

(Stratix V - 5SGXMABN3F45I4) – (Sem restrições de síntese)

6.1.4 DCT 32x32

Para a DCT 32x32, ocorre os mesmos procedimentos descritos para as DCTs de menores tamanhos. A estrutura da DCT de 32 pontos, que está representada na Figura 6.6, também foi replicada com as devidas alterações, para gerar a 2ª DCT 1-D e, entre elas, é inserida a matriz de transposição, com tamanho 32x32. Novamente, esta arquitetura reusa a arquitetura anterior, neste caso, a arquitetura da transformada 16x16.

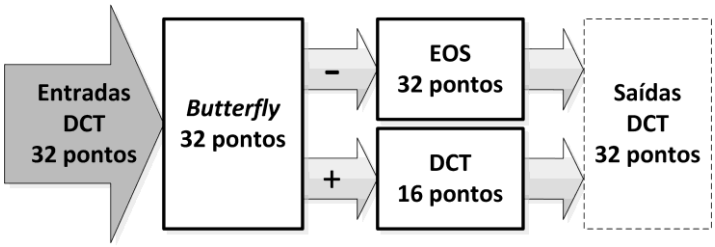

Figura 6.6 – Arquitetura 1-D da DCT 32x32

Os dados gerados através da ferramenta de síntese após a descrição em VHDL das novas equações otimizadas geradas pelo software e de otimização (Apêndice A) estão descritos na Tabela 6.4.

Tabela 6.4 – Dados de síntese da DCT 32x32

DCT 32x32	Versão Otimizada			
	1ª DCT 1-D	Matriz de Transposição	2ª DCT 1-D	DCT 2-D
ALMs	4.058	*	*	17.438
Registradores	0	*	*	17.867
Frequência	67,44 MHz	*	*	58,98 MHz

(Stratix V - 5SGXMABN3F45I4) – (Sem restrições de síntese)
*(Pinos Insuficientes)

A arquitetura da DCT 32x32 consome 32 amostras por ciclo de *clock*, que é o máximo paralelismo possível por conta do uso da separabilidade.

6.1.5 DCT 2-D de Múltiplos Tamanhos

Por fim, foi criada uma arquitetura DCT 2-D de múltiplos tamanhos, capaz de realizar os cálculos de qualquer uma das DCTs presentes no HEVC.

Como já foi explicado anteriormente, uma DCT é parte integrante de outra de maior tamanho. Desta forma, para a implementação da transformada DCT 2-D de múltiplos tamanhos, foi criada uma arquitetura baseada na DCT 32x32. Uma vez que as demais DCTs fazem parte internamente desta DCT e por sua vez compartilham o mesmo hardware, através de algumas alterações que basicamente se resumem à inserção de multiplexadores para desviarem as entradas e saídas para os locais corretos, é possível configurar o hardware para realizar os cálculos da DCT do tamanho escolhido.

Com as estruturas das DCTs de diferentes tamanhos já finalizadas, a implementação da DCT 2-D de múltiplos tamanhos foi realizada de forma bastante simples, com poucas modificações.

A Figura 6.7 mostra a DCT 1-D de múltiplos tamanhos, onde podem ser percebidas as ligações de entrada e saída necessárias para utilizar cada tamanho de DCT. Nesta figura, é possível notar que as entradas para as DCTs de 4, 8 e 16 pontos são inseridas no bloco *butterfly*, respectivamente, dentro da arquitetura geral. Assim, quando a DCT de determinado tamanho é selecionada pelo controle geral, as entradas devem ser ligadas em suas respectivas entradas do bloco *butterfly*. Isto é realizado através da inserção de multiplexadores nas entradas destes blocos.

As saídas das DCTs 1-D de todos os tamanhos passam por uma etapa final de arredondamento que utiliza as variáveis *add* e *shift* (Tabela 4.2). Os valores destas variáveis são dependentes tanto do tamanho da DCT como se esta é a 1ª DCT 1-D ou a 2ª DCT 1-D.

Para cada tamanho de transformada um conjunto diferente de saídas da arquitetura é válido. Considerando a Figura 6.7, para a transformada de 32 pontos, todas as saídas (Saída 1 a Saída 32) são utilizadas. Já para a transformada de 16 pontos, apenas 16 saídas são válidas (Saída 1 à Saída

16). Para a transformada de 8 pontos, são 8 as saídas válidas (Saída 1 à Saída 8). Finalmente, a transformada de 4 pontos terá quatro saídas válidas (Saída 1 à Saída 4). A arquitetura da DCT 2-D de múltiplos tamanhos utiliza duas instâncias de arquiteturas apresentadas na Figura 6.7 e uma matriz de transposição.

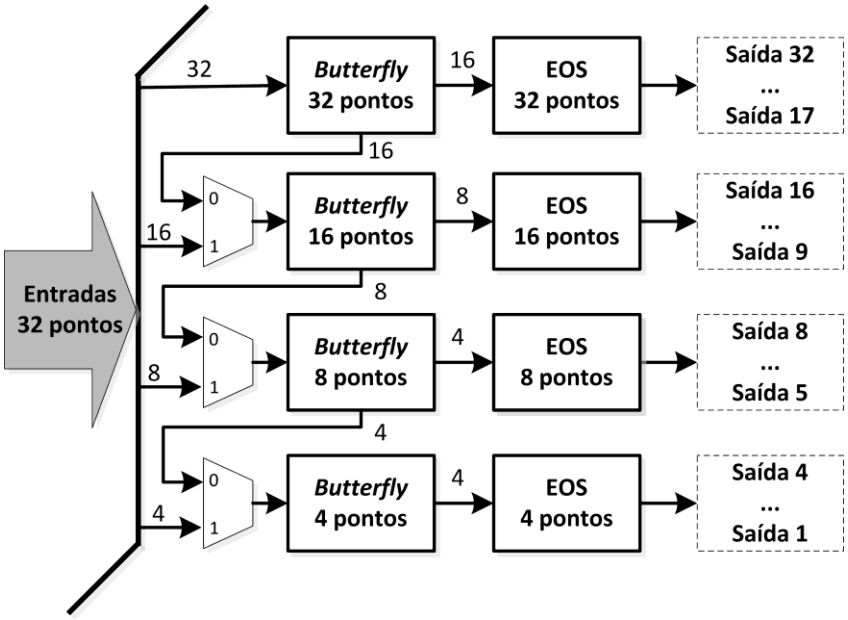

Figura 6.7 – Arquitetura da transformada DCT 2-D de múltiplos tamanhos

A matriz de transposição utilizada para ligar as duas arquiteturas DCT 1-D de múltiplos tamanhos foi implementada utilizando-se apenas uma matriz de registradores, em que os registradores são adequadamente ligados visando permitir a transposição, sem qualquer atraso de tempo nas arquiteturas das DCTs 1-D. As entradas da matriz de transposição são as saídas das DCTs de diferentes tamanhos e as saídas da matriz de transposição são as entradas transpostas para as 2^{aS} DCTs 1-D de diferentes tamanhos. A depender do tamanho da DCT definida pelo sinal de controle, a cada 4, 8, 16 ou 32 ciclos de *clock*, os sinais dos multiplexadores também são alterados e assim a matriz inverte o processo de leitura e escrita de linha para coluna e vice-versa. Desta forma, é possível efetuar a leitura e a escrita simultaneamente na mesma matriz.

O nível de paralelismo da DCT 2-D de múltiplos tamanhos depende do tamanho selecionado de DCT, sendo o máximo paralelismo permitido para

aquele tamanho de transformada. Assim, dependendo da entrada de controle, a arquitetura pode processar 4, 8, 16 ou 32 amostras por ciclo.

Os dados de síntese da arquitetura da DCT 2-D de Múltiplos Tamanhos estão presentes na Tabela 6.5.

Tabela 6.5 – Dados de síntese da DCT 2-D de Múltiplos Tamanhos

Multi-DCT	Versão Otimizada			
	1ª DCT 1-D	Matriz de Transposição	2ª DCT 1-D	DCT 2-D
ALMs	4.534	*	*	19.100
Registradores	0	*	*	16.403
Frequência*	63,95 MHz	*	*	55,36 MHz

(Stratix V - 5SGXMABN3F45I4) – (Sem restrições de síntese)
*(Pinos Insuficientes)

Como pode ser percebido, os dados de síntese são próximos daqueles obtidos na arquitetura implementada da DCT 32x32. Desta forma, esta arquitetura de DCT de Múltiplos Tamanhos se mostra extremamente vantajosa pois é capaz de realizar qualquer uma das DCTs de diferentes tamanhos com praticamente os mesmos recursos de hardware da DCT 32x32.

6.2 Análise dos Dados e Resultados de Síntese

A Tabela 6.6, a Tabela 6.7, a Tabela 6.8, e a Tabela 6.9 apresentam análises comparativas com os dados de síntese das DCTs de diferentes tamanhos das arquiteturas implementadas, considerando diferentes configurações da ferramenta de síntese. Nestas tabelas constam a frequência máxima de operação das arquiteturas, o número de módulos lógicos adaptativos (ALMs) e blocos DSPs utilizados do FPGA alvo.

A ferramenta de síntese Quartus II possui algoritmos de otimização muito avançados. Ela é capaz de interpretar as diferentes arquiteturas descritas e realizar otimizações muito eficientes. O Quartus II utiliza muitas das técnicas conhecidas e descritas na literatura para realizar estas otimizações. Por isso, como este trabalho envolve otimizações de equações visando implementações em hardware, foi considerado relevante a realização de comparações entre os resultados da arquitetura descrita com base nas otimizações propostas e os

resultados gerados pela ferramenta através da simples descrição das equações originais do HEVC sem nenhum processo de otimização anterior à síntese no Quartus II.

Foram sintetizados e gerados dados para todas as transformadas DCTs de diferentes tamanhos tanto para a solução proposta otimizada como para a versão com as equações originais.

Além disso, foram realizadas quatro diferentes sínteses para todas as arquiteturas, com habilitação de diferentes formas de otimização da ferramenta de síntese. Na primeira síntese nenhuma restrição foi imposta para a ferramenta. Os dados gerados nesta síntese estão descritos na Tabela 6.6. Não foi possível gerar os dados da DCT 32x32 da versão com equações originais, pois a síntese excedeu os recursos de hardware disponíveis no FPGA escolhido, que é o maior da família.

Os dados de síntese da versão original descrita no HEVC e da versão otimizada proposta neste trabalho estão referenciadas nas tabelas a seguir como “Orig.” e “Otim.”, respectivamente.

Tabela 6.6 - Síntese sem restrições

DCT			4x4		8x8		16x16		32x32	
			Orig.	Otim.	Orig.	Otim.	Orig.	Otim.	Orig.	Otim.
1ª DCT 1D	ALMs	Total	89	128	262	454	723	1.304	2.077	4.058
		Ganho	- 43,82%		- 73,28%		- 80,36%		- 95,38%	
	Reg		0	0	0	0	0	0	0	0
	DSPs		2	0	12	0	54	0	249	0
	Freq. (MHz)	Total	131,42	128,92	69,54	93,65	37,04	79,11	17,34	67,44
		Ganho	- 1,90%		34,67%		113,58%		288,93%	
Matriz	ALMs	Total	151		573		2.175		*	
	Reg		269		1.093		4.413		*	
	DSPs		0		0		0		*	
	Freq. (MHz)		580,05		580,05		423,37		*	
2ª DCT 1D	ALMs	Total	171	210	413	672	1049	1903	*	*
		Ganho	- 22,81%		- 62,71%		- 81,41%		-	
	Reg		0	0	0	0	0	0	*	*
	DSPs		2	0	12	0	54	0	*	*
	Freq. (MHz)	Total	118,76	114,05	63,76	80,72	34,31	71,43	*	*
		Ganho	- 3,97%		26,60%		108,19%		-	
DCT 2D	ALMs	Total	394	471	1.162	1.622	3.795	5.256	**	17.438
		Ganho	- 19,54%		-39,59%		- 38,50%		-	
	Reg		292	283	1.154	1.130	4.412	4.437	**	17.867
	DSPs		4	0	24	0	108	0	**	0
	Freq. (MHz)	Total	104,62	108,25	63,61	78,97	35,04	69,80	**	58,98
		Ganho	3,47%		24,15%		99,20%		-	

(Stratix V - 5SGXMABN3F45I4)

*(Pinos Insuficientes) - **(DSPs Insuficientes)

Como pode ser percebido, para as arquiteturas descritas a partir das equações originais do HEVC, o Quartus II utilizou blocos DSPs dedicados para implementar os multiplicadores. Desta forma, o número de ALMs usadas para estas arquiteturas se mostra mais reduzido quando comparado com as arquiteturas otimizadas. Mas é importante perceber que nenhum bloco DSP foi usado na versão otimizada. Além disso, as frequências de operação para as arquiteturas originais se mostraram muito inferiores quando comparadas com as versões otimizadas para blocos maiores. Esta diferença fica mais evidenciada quanto maior for o tamanho da DCT. A única arquitetura que se mostrou, parcialmente, pior quanto à frequência foi a DCT 4x4 que apresentou aproximadamente 4MHz a menos que a versão com as equações originais para a 2ª DCT 1D e um diferença menor que 3MHz para a 1ª DCT 1D. Neste caso, de menor complexidade, as otimizações do Quartus II foram mais eficientes, o que não aconteceu para transformadas maiores.

Os ganhos relacionados à frequência de operação se mostraram expressivos especialmente para as DCTs maiores. Por exemplo, na DCT 1-D de tamanho 32 os ganhos em frequência de operação se mostram superiores a 288%.

Na segunda análise foi desabilitado o recurso de otimização interna na ferramenta de síntese. Para tanto, basicamente é necessário inserir após a declaração dos sinais internos, as seguintes linhas de código:

- signal keep_wire : std_logic;
- attribute keep: boolean;
- attribute keep of <lista de sinais, entradas e saidas>: signal is true;

Esta configuração evita que a ferramenta otimize os caminhos de dados definidos. Os dados de síntese para esta configuração estão apresentados na Tabela 6.7.

Na Tabela 6.7, é possível perceber que para ambas arquiteturas, original e otimizada, para todos os tamanhos de transformadas, as frequências de operação pioraram, assim como o uso de ALMs também aumentou. Isto é explicado pelo fato desta configuração de síntese impedir otimizações internas.

Tabela 6.7 - Dados de síntese com DSP e sem otimizações internas

DCT			4x4		8x8		16x16		32x32	
			Orig.	Otim.	Orig.	Otim.	Orig.	Otim.	Orig.	Otim.
1ª DCT 1D	ALMs	Total	239	313	545	823	1.279	2.343	3.394	8.155
		Ganho	- 30,96%		- 51,01%		- 83,19%		- 140,28%	
	Reg		0	0	0	0	0	0	0	0
	DSPs		2	0	12	0	54	0	249	0
	Freq. (MHz)	Total	110,86	107,99	61,18	88,22	33,56	71,54	16,55	58,35
		Ganho	- 2,59%		44,20%		113,17%		252,57%	
Matriz	ALMs	Total	168		583		2.183		*	
	Reg		273		1.085		4.349		*	
	DSPs		0		0		0		*	
	Freq. (MHz)		580,05		580,05		455,17		*	
2ª DCT 1D	ALMs	Total	318	414	748	1.166	1.791	3.416	*	*
		Ganho	- 30,19%		- 55,88%		- 90,73%		-	
	Reg		0	0	0	0	0	0	*	*
	DSPs		2	0	12	0	54	0	*	*
	Freq. (MHz)	Total	104,54	99,71	58,26	75,75	33,53	63,70	*	*
		Ganho	- 4,62%		30,02%		89,98%		-	
DCT 2D	ALMs	Total	723	894	1.875	2.569	5.251	7.949	**	26.680
		Ganho	- 23,65%		- 37,01%		- 51,38%		-	
	Reg		273	269	1.121	1.109	4.372	4.384	**	17.848
	DSPs		4	0	24	0	108	0	**	0
	Freq. (MHz)	Total	88,98	92,72	52,73	73,87	31,81	61,11	**	48,90
		Ganho	4,20%		40,09%		92,11%		-	

(Stratix V - 5SGXMABN3F45I4)
*(Pinos Insuficientes) - **(DSPs Insuficientes)

Assim como na síntese anterior (Tabela 6.6), a versão otimizada não utiliza blocos DSPs e continua apresentando expressivos ganhos em frequência de operação quando comparada com a versão original. Sendo que para esta configuração inclusive para a DCT 4x4 a versão otimizada apresenta uma frequência de operação mais elevada que a versão original e o ganho quanto à frequência de operação para a DCT de 32 pontos se mantêm elevado, com ganho superior a 252%.

Para a terceira síntese foi desabilitado o uso dos blocos DSPs. Para tanto, devem ser seguidos os passos a seguir:

Configurar blocos DSP (com o projeto criado): Assignments -> Settings/ Analysis & Synthesis Settings/ More Settings...:

- Setar para "Off" a opção "Auto DSP Block Replacement"
- Setar para "Off" a opção "DSP Block Balancing"
- Setar para "0" a opção "Maximum DSP Block Usage"

Para esta síntese, foram desabilitados os blocos DSPs e mantido habilitado o recurso de otimização interna na ferramenta de síntese. Os resultados estão mostrados na Tabela 6.8.

Na tabela Tabela 6.8, é possível perceber que sem o uso dos blocos de DSPs, as frequências de operação das arquiteturas da versão com equações originais se mostraram melhores do que na síntese anterior, quando estavam utilizando os blocos DSPs. Mas mesmo assim as arquiteturas otimizadas acabam por mostrar, na maioria dos casos, frequências ainda superiores, especialmente para transformadas de tamanhos maiores (16x16 e 32x32).

Mas nesta síntese, o que mais se destaca é a diferença entre o número de ALMs utilizados entre as diferentes arquiteturas, onde ficam bem claros os ganhos gerados pelas otimizações propostas neste trabalho. Em todos os casos as otimizações propostas apresentam ganhos em relação ao uso das equações originais. Aqui também os ganhos mais expressivos se mostram mais evidentes nas DCTs de tamanhos maiores.

Tabela 6.8 – Dados de síntese sem DSP e com otimizações internas

DCT			4x4		8x8		16x16		32x32	
			Orig.	Otim.	Orig.	Otim.	Orig.	Otim.	Orig.	Otim.
1ª DCT 1D	ALMs	Total	130	128	564	454	2.053	1.304	7.500	4.058
		Ganho	1,54%		19,50%		36,48%		45,89%	
	Reg		0	0	0	0	0	0	0	0
	DSPs		0	0	0	0	0	0	0	0
	Freq. (MHz)	Total	144,51	128,92	107,45	93,65	76,61	79,11	52,66	67,44
		Ganho	- 10,79%		- 12,84%		3,26%		28,07%	
Matriz	ALMs	Total	151		573		2.175		*	
	Reg		269		1.093		4.413		*	
	DSPs		0		0		0		*	
	Freq. (MHz)		580,05		580,05		423,37		*	
2ª DCT 1D	ALMs	Total	258	210	1016	672	3681	1903	*	*
		Ganho	18,60%		33,86%		48,30%		-	
	Reg		0	0	0	0	0	0	*	*
	DSPs		0	0	0	0	0	0	*	*
	Freq. (MHz)	Total	113,64	114,05	90,44	80,72	63,75	71,43	*	*
		Ganho	0,36%		- 10,75%		12,05%		-	
DCT 2D	ALMs	Total	527	471	2070	1622	7779	5256	28656	17438
		Ganho	10,63%		21,64%		32,43%		39,15%	
	Reg		290	283	1.131	1.130	4.371	4.437	17800	17.867
	DSPs		0	0	0	0	0	0	0	0
	Freq. (MHz)	Total	103,83	108,25	81,76	78,97	63,33	69,80	45,36	58,98
		Ganho	4,26%		- 3,41%		10,22%		30,03%	

(Stratix V - 5SGXMABN3F45I4)
*(Pinos Insuficientes)

Como última síntese foi desabilitado o uso de blocos DSP e também o recurso de otimização interna na ferramenta de síntese. Os dados desta síntese estão expressos na Tabela 6.9. Nesta síntese, como nas demais, as arquiteturas implementadas com as equações otimizadas pelo software desenvolvido neste trabalho se mostram, na maioria dos casos, melhores quando comparados com as versões com equações originais. Em todos os casos existe ganhos do ponto de vista do consumo de recursos. Por outro lado, em alguns casos há uma pequena redução na frequência máxima atingida.

É preciso salientar que analisando apenas as DCTs de maior tamanho, ou seja, DCT 16x16 e DCT 32x32, em nenhuma das diferentes configurações de síntese as versões otimizadas perderam para a versão com equações originais. Em todos os casos, a versão otimizada sempre mostrou ganhos expressivos ou em redução ao uso de ALMs ou quanto à frequência máxima de operação atingida.

Tabela 6.9 - Dados de síntese sem DSP e sem otimizações internas

DCT			**4x4**		**8x8**		**16x16**		**32x32**	
			Orig.	Otim.	Orig.	Otim.	Orig.	Otim.	Orig.	Otim.
1ª DCT 1D	ALMs	Total	279	313	845	823	2589	2343	8646	8155
		Ganho	- 12,19%		2,60%		9,50%		5,68%	
	Reg		0	0	0	0	0	0	0	0
	DSPs		0	0	0	0	0	0	0	0
	Freq. (MHz)	Total	121,95	107,99	92,64	88,22	63,19	71,54	51,09	58,35
		Ganho	- 11,45%		- 4,77%		13,21%		14,21%	
Matriz	ALMs	Total	168		583		2.183		*	
	Reg		273		1.085		4.349		*	
	DSPs		0		0		0		*	
	Freq. (MHz)		580,05		580,05		455,17		*	
2ª DCT 1D	ALMs	Total	405	414	1345	1166	4388	3416	*	*
		Ganho	- 2,22%		13,31%		22,15%		-	
	Reg		0	0	0	0	0	0	*	*
	DSPs		0	0	0	0	0	0	*	*
	Freq. (MHz)	Total	95,28	99,71	77,98	75,75	61,03	63,70	*	*
		Ganho	4,65%		- 2,86%		4,37%		-	
DCT 2D	ALMs	Total	850	894	2771	2569	9160	7949	31495	26680
		Ganho	- 5,18%		7,29%		13,22%		15,29%	
	Reg		268	269	1.115	1.109	4.359	4.384	17720	17.848
	DSPs		0	0	0	0	0	0	0	0
	Freq. (MHz)	Total	94,49	92,72	76,02	73,87	57,00	61,11	42,09	48,90
		Ganho	- 1,87%		- 2,83%		7,21%		16,18%	

(Stratix V - 5SGXMABN3F45I4)

*(Pinos Insuficientes)

Como o objetivo da implementação do software foi justamente realizar as otimizações para as DCTs de maiores tamanhos devido à elevada complexidade envolvida nestas transformadas, pode-se considerar que as otimizações desenvolvidas neste trabalho atingiram seus objetivos.

Por fim, a Tabela 6.10 apresenta o número de quadros por segundo que cada arquitetura otimizada 2-D é capaz de processar, tomando como base os dado de síntese da Tabela 6.6 (síntese sem restrição), considerando duas resoluções: FHD (1920x1080 *pixels*) e QFHD (3840x2160 *pixels*). As taxas de quadro consideram uma taxa de sub-amostragem de 4:2:0 (RICHARDSON, 2003). Também é apresentada a frequência mínima necessária para atingir 30 quadros por segundo para cada arquitetura. Para gerar esta tabela foram usados os resultados de síntese apresentados na Tabela 6.6.

Tabela 6.10 – Número de quadros por segundo e frequência mínima para 30 quadros por segundo

DCT 2-D	FHD fps	FM 30fps* FHD (MHz)	QFHD fps	FM 30fps* QFHD (MHz)
4x4	139,21	23,33	34,80	93,31
8x8	203,11	11,66	50,77	46,66
16x16	359,05	5,83	89,76	23,33
32x32	606,79	2,91	151,69	11,66

* Frequência mínima para processar 30 quadros por segundo (Dados de Síntese sem restrição)

Como são necessários no mínimo 30 quadros por segundo para gerar a sensação de movimento contínuo, todas as arquiteturas se mostram capazes de processar vídeos digitais de altíssima resolução em tempo real.

O motivo pelo qual as arquiteturas de menor tamanho processam menos quadros por segundo que as DCTs de maiores tamanhos, se deve ao fato que a DCT 32x32, por exemplo, processa em paralelo 32 amostras por ciclo de *clock*, já a arquitetura da DCT 4x4 é capaz de processar apenas 4 amostras por ciclo.

A depender da taxa de processamento que se queira atingir, é possível instanciar mais de uma DCT de mesmo tamanho para aumentar ainda mais o desempenho. Por exemplo, podem ser instanciadas quatro DCTs 4x4 e, deste modo, seriam processadas 16 amostras por ciclo.

Através dos dados apresentados na tabela Tabela 6.10 é possível constatar que todas as arquiteturas implementadas das DCTs de diferentes tamanhos alcançam elevadas taxas de processamento, mesmo se tratando da resolução QFHD. Todas implementações são capazes de processar mais de 120 fps para a resolução FHD e mais de 30 fps para a resolução QFHD.

6.2.1 Resultados da DCT 2-D de Múltiplos Tamanhos

Em função das especificidades da DCT 2-D de múltiplos tamanhos, os seus resultados serão discutidos separadamente.

Inicialmente, foram geradas arquiteturas da DCT 1-D de múltiplos tamanhos, considerando as equações originais do HM e as equações otimizadas pelo software desenvolvido neste trabalho.

Os resultados de síntese da Multi-DCT são apresentados na Tabela 6.11. Os principais resultados apresentados são: a frequência máxima de operação e o número de módulos lógicos adaptativos (ALMs). Para a Multi-DCT não foi possível gerar dados da matriz intermediária, bem como para a 2ª DCT 1-D, por que estes dois módulos excedem o número de pinos de entrada e saída do dispositivo FPGA.

A partir dos resultados apresentados na Tabela 6.11, fica claro que as equações otimizadas alcançam ganhos expressivos em termos de máxima frequência de operação e, principalmente, na utilização de recursos de hardware. A redução em termos de recursos de hardware na DCT 1-D foi superior a 43% e o aumento da frequência de operação foi superior a 19% quando comparado com os dados de síntese sem nenhuma restrição.

Estes ganhos são resultados das otimizações propostas, uma vez que as implementações de ambas arquiteturas foram realizadas com a mesma ferramenta de síntese e com as mesmas configurações. Neste caso, a síntese foi configurada para não usar blocos DSP e permitir otimizações internas nos caminhos de dados.

A arquitetura da DCT 2-D de múltiplos tamanhos é composta por duas DCTs 1-D de múltiplos tamanhos e a matriz de transposição.

Tabela 6.11 – Resultados de Síntese da DCT de múltiplos tamanhos

Multi-DCT			Com DSP Com Otim.		Com DSP Sem Otim.		Sem DSP Com Otim.		Sem DSP Sem Otim.	
			Orig.	Otim.	Orig.	Otim.	Orig.	Otim.	Orig.	Otim.
1ª DCT 1D	ALMs	Total	2625	4534	3968	8729	7962	4534	9222	8729
		Ganho	- 72,72%		- 119,98%		43,05%		5,35%	
	Reg		0	0	0	0	0	0	0	0
	DSPs		249	0	249	0	0	0	0	0
	Freq. (MHz)	Total	17,84	63,95	17,15	55,16	53,68	63,95	48,74	55,16
		Ganho	258,46%		221,63%		19,13%		13,17%	
DCT 2D	ALMs	Total	***	19100	***	28509	30013	19100	32867	28509
		Ganho	-		-		36,36%		13,26%	
	Reg		***	16403	***	16396	16414	16403	16396	16396
	DSPs		***	0	***	0	0	0	0	0
	Freq. (MHz)	Total	***	55,36	***	44,28	44,25	55,36	40,76	44,28
		Ganho	-		-		25,11%		8,64%	

Os resultados da DCT 2-D de múltiplos tamanhos mostram a relevância das otimizações propostas, uma vez que a frequência de operação aumentou em mais de 25% e a utilização de ALMs foi reduzida em mais de 36%. Isto significa que a DCT 2-D de múltiplos tamanhos apresenta expressivos ganhos quanto a custo de hardware e frequência de operação, quando comparada com a solução não otimizada.

Para todos os cálculos de quadros por segundo apresentados neste trabalho, não foi considerado o *loop* de processamento que o codificador realiza.

Além disso, é importante destacar que este hardware é capaz de realizar todos os cálculos das DCTs de diversos tamanhos definidas no HEVC. Assim, seria possível usar apenas uma arquitetura otimizada, evitando o uso de quatro diferentes arquiteturas de DCTs 2-D, atingindo uma redução expressiva no uso de hardware para este módulo do codificador.

A Tabela 6.12 mostra as taxas de processamento alcançadas pela arquitetura otimizada da DCT 2-D de múltiplos tamanhos, considerando o tamanho da DCT selecionada (4x4, 8x8, 16x16 e 32x32). Os resultados apresentados são o número de quadros processados por segundo (fps), considerando duas resoluções: FHD e QFHD e as frequências mínimas de operação necessárias para que a arquitetura atinja o processamento de 30 quadros por segundo. É importante notar que as DCTs de menores tamanhos apresentam taxas de processamento menores porque o número de amostras processadas por ciclo varia de acordo com o tamanho da DCT e a frequência

de operação é constante. Por exemplo, o tamanho DCT 4x4 da DCT 2-D de múltiplos tamanhos consumirá quatro amostras por ciclo. Por outro lado, a mesma arquitetura configurada para a DCT 32x32 irá processar 32 amostras por ciclo.

Tabela 6.12 – Taxa de Processamento

DCT 2-D	FHD fps	FM 30fps* FHD (MHz)	QFHD fps	FM 30fps* QFHD (MHz)
4x4	71,19	23,33	17,79	93,31
8x8	142,38	11,66	35,59	46,66
16x16	284,77	5,83	71,19	23,33
32x32	569,54	2,91	142.38	11,66

* Frequência mínima para processar 30 quadros por segundo
(Dados de Síntese sem restrição)

Os resultados apresentados na Tabela 6.12 demonstram que a DCT 2-D de múltiplos tamanhos implementada em hardware é capaz de atingir mais do que 60 quadros por segundo para as DCTs de diferentes tamanhos, quando a resolução FHD é considerada. Considerando a resolução QFHD, a DCT 2-D de múltiplos tamanhos é capaz de atingir mais de 30 quadros por segundo para quase todos os tamanhos de DCT. Apenas para a DCT 4x4 a arquitetura não é capaz de atingir 30 quadros por segundo.

6.3 Comparações com Trabalhos Relacionados

Uma vez que o HEVC ainda está em fase de desenvolvimento, existem poucos trabalhos publicados sobre implementações de hardware da transformada DCT para este padrão. Os poucos trabalhos publicados sobre este assunto, estão se concentrando em diferentes tecnologias, o que impede uma comparação justa. A Tabela 6.13 mostra um resumo com os resultados dos trabalhos encontrados na literatura que abordam as transformadas DCT no HEVC.

Os dados apresentados na Tabela 6.13 na coluna “Este trabalho” correspondem aos dados de síntese sem nenhuma restrição da Multi-DCT. Para que fossem realizadas comparações mais justas, cada trabalho deveria ser comparado com a arquitetura correspondente apresentada neste trabalho e

todos sintetizados com a mesma tecnologia. Mesmo assim através da Tabela 6.13 é possível realizar algumas comparações, como descritas a seguir.

Tabela 6.13 – Comparação com trabalhos anteriores

		Shen	Budagavi	Ahmed	Edirisuriya		Martuza	Este Trabalho
DCT 4x4		✓						✓
DCT 8x8		✓					✓	✓
DCT 16x16		✓		✓	✓			✓
DCT 32x32		✓	✓					✓
Tecnologia		st.-cells 0.13µm	st.-cells 0.45nm	st.-cells 0.90nm	FPGA Virtex-6	st.-cells 0.45nm	st.-cells 0.18µm	FPGA Stratix-V
Amostras por ciclo		4	32	16	16		1	4/8/16/32
1D	Área			0,290 mm^2				
	ALM							4534
	Reg.							0
	Gate count	109.20 K	148.00 K				12.30 K	
	Freq.	350 MHz	250 MHz	150 MHz			211 MHZ	64 MHZ
2D	Área					0,195 mm^2		
	Flip-flop				5454			
	LUTs				4724			
	Slice				1371			
	ALM							19.100
	Reg.							16.403
	Freq.				250 MHz	900 MHz		55,36

Shen et al. (2012) apresenta um projeto de hardware para a DCT de diferentes tamanhos presentes em codificadores de vídeos. Este trabalho utiliza memória em vez de registradores para implementar a matriz de transposição. A mesma arquitetura DCT 1-D é reutilizada para fazer o cálculo da 2-D e assim, a taxa de processamento da DCT 2-D é prejudicada. Além disso, Shen mostra que a técnica MCM (*Multiple Constant Multiplication*) não é tão vantajosa para as DCTs de maiores tamanhos. Assim, ele usa um multiplicador comum para a DCT de 16 e 32 pontos. Finalmente, a arquitetura de Shen é capaz de processar quatro amostras de entrada por ciclo de clock, e foi concebido em cinco estágios de *pipeline*, atingindo uma frequência de operação de 191MHz. Este trabalho foi implementado na tecnologia *standard-cells*-0,13um. Nenhuma otimização algorítmica é apresentada. A arquitetura apresentada neste trabalho de mestrado, além de ser baseada em otimizações algorítmicas, é capaz de atingir uma taxa de processamento maior do que a proposta por Shen, em especial para as transformadas de tamanhos maiores.

Budagavi et al. (2012) propôs uma solução de hardware para a DCT de 32 pontos. Esta arquitetura foi descrita em RTL (*Register Transfer Level*) e sintetizada para uma tecnologia *standard-cells*-45nm. Esta arquitetura é capaz de realizar seus cálculos em uma frequência de 250 MHz e processar 32 amostras por ciclo de *clock*. Desta forma, este trabalho apresenta as maiores taxas de processamento entre todos os trabalhos relacionados encontrados até o momento. Por outro lado, apenas um tamanho de transformada é suportado e o custo de hardware para suportar o paralelismo usado, sem qualquer otimização algorítmica é muito alto e mesmo usando tecnologias diferentes, é possível inferir que a solução proposta neste trabalho, utilizaria muito menos recursos de hardware se projetada na mesma tecnologia e suportando apenas um tamanho de transformada.

O trabalho apresentado em (AHMED et al., 2011) mostra uma solução arquitetural para a DCT de 16 pontos. Foi implementada em tecnologia *standard-cells*-90nm. Este hardware opera com uma frequência de 150 MHz, e é capaz de processar 16 amostras por ciclo. Mais uma vez, um único tamanho de transformada é suportado e o custo do hardware para suportar o paralelismo usado sem qualquer otimização algorítmica tende a ser mais elevado que o hardware desenvolvido neste trabalho.

Edirisuriya et al. (2012) propõe uma arquitetura para um mecanismo multi-transformada capaz de processar a DCT/DST 2-D 16x16 sem o uso de multiplicadores. Esta arquitetura trabalha a uma frequência de 250 MHz e processa 16 amostras por ciclo de *clock*. A arquitetura proposta foi implementada usando um FPGA e também mapeada para uma tecnologia *standard-cells*-45nm. Mais uma vez, o alto nível de paralelismo sem otimizações algorítmicas implica um custo elevado de hardware.

Em (MARTUZA et al, 2012) é apresentada uma nova arquitetura de hardware compartilhada para calcular as IDCTs 8x8 do HEVC e do H.264/AVC. Sua arquitetura é capaz de trabalhar em 211,4 MHz e processa uma amostra por ciclo de *clock*. O hardware foi descrito em Verilog e foi sintetizado em um FPGA Xilinx Virtex4. A arquitetura proposta nesta dissertação é capaz de processar quatro tamanhos de DCT, utilizando o mesmo hardware, e é capaz de atingir taxas de processamento mais elevadas do que a proposta

apresentada em (MARTUZA et al, 2012) para todos os tamanhos de transformada.

Uma arquitetura de DCT de 16 pontos foi desenvolvida no escopo deste trabalho antes da aplicação das heurísticas de otimização já apresentadas, mas realizando algumas otimizações manualmente. Este trabalho está relatado em (JESKE et al., 2012). A arquitetura é capaz de processar 16 amostras por ciclo de *clock* e foi sintetizada para dois FPGAs diferentes. O hardware implementado de forma puramente combinacional e algumas otimizações algorítmicas foram implementadas. A maior frequência de operação foi de 87,60 MHz, atingindo uma taxa de processamento competitiva. O trabalho, apresentado nesta dissertação, é capaz de processar múltiplos tamanhos de transformada discreta do cosseno e as otimizações algorítmicas foram mais eficazes para permitir uma menor utilização de recursos de hardware.

7 CONCLUSÕES

O padrão HEVC de codificação de vídeo foi desenvolvido para suprir a crescente demanda por suporte a elevadas resoluções de vídeos em uma diversidade enorme de dispositivos e aplicações. Assim, taxas de compressão mais elevadas estão sendo exigidas pelo mercado, sempre mantendo a qualidade do vídeo. Deste modo, investigações relacionadas com otimizações algorítmicas compatíveis com o padrão e com o desenvolvimento arquitetural para as ferramentas de codificação deste padrão é uma atividade de grande relevância no cenário atual.

Este trabalho teve como objetivo desenvolver otimizações algorítmicas focadas em implementações em hardware das transformadas DCT 2-D de vários tamanhos definidas pelo HEVC, bem como o desenvolvimento do hardware para estes algoritmos otimizados.

Como as maiores contribuições que este trabalho apresentou, destacam-se a abordagem das técnicas de otimização de forma conjunta e não sequencialmente, como realizado usualmente em outros trabalhos, e a implementação de um algoritmo capaz de realizar as análises nas diferentes implementações possíveis, buscando a melhor solução, de forma totalmente automatizada, que resulta em um menor hardware. Além disso, foram geradas arquiteturas de elevada taxa de processamento considerando os algoritmos otimizados.

Para tanto, inicialmente realizou-se um estudo sobre as diferentes ferramentas propostas para o novo padrão de codificação de vídeos HEVC, bem como uma comparação com as ferramentas já existentes no atual padrão de codificação H.264/AVC. Nesta etapa o trecho de código relacionado com as transformadas no software de referência do HEVC (HM) foi identificado.

Então, os algoritmos originais das DCTs de vários tamanhos propostas no HEVC foram analisados e diversas simplificações foram propostas visando uma eficiente implementação em hardware. As multiplicações foram substituídas por somas de deslocamentos e subexpressões comuns foram compartilhadas para reduzir o custo de hardware. A abordagem adotada na otimização é uma contribuição deste trabalho, já que tanto o processo de transformação de multiplicações em somas de deslocamentos quanto o processo de compartilhamento de subexpressões comuns foram realizados simultaneamente, gerando uma solução com maior otimização do que se as duas etapas fossem executadas sequencialmente, com otimizações locais. Como o conjunto de combinações a serem avaliadas era proibitivamente grande, foi desenvolvido um software para automatizar este processo.

As otimizações propostas permitiram, no melhor caso (DCT 32x32) uma redução expressiva no número de operações aritméticas de 22 mil multiplicações e 25 mil somas ou subtrações para 39 mil somas ou subtrações, considerando apenas um bloco de 32x32 amostras. Analisando as somas ou subtrações de um *bit* utilizadas, foi possível gerar um ganho superior a 59% se comparado com a versão original do HEVC.

Os algoritmos otimizados foram usados como base para o desenvolvimento arquitetural. Foram cinco as arquiteturas de DCTs 2-D desenvolvidas, uma para cada tamanho de transformada e uma capaz de processar todos os tamanhos de transformada. As arquiteturas de DCT 2-D foram desenvolvidas usando duas arquiteturas de DCT 1-D e uma matriz de transposição. As arquiteturas de DCT 1-D foram desenvolvidas de forma puramente combinacional, com o paralelismo máximo permitido para cada tamanho de transformada (4, 8, 16 ou 32 amostras por ciclo).

Os resultados de síntese das DCTs de diferentes tamanhos, mostraram que as arquiteturas desenvolvidas são capazes de processar vídeos de resolução muito elevada, como QFHD (3840x2160 *pixels*) em tempo real. Versões arquiteturais focadas no algoritmo não otimizado também foram desenvolvidas para avaliar os ganhos que as simplificações no algoritmo geraram no projeto de hardware.

Assim, com os resultados de otimização algorítmica e de desenvolvimento arquitetural, é possível concluir que a abordagem das

técnicas de otimizações analisadas em conjunto, bem como o algoritmo implementado para realizar as buscas se mostraram eficientes, atingindo seus objetivos, pois foi possível a geração de um hardware extremamente otimizado e com elevada taxa de processamento para o módulo das transformadas do HEVC.

Como trabalhos futuros pretende-se explorar diferentes versões arquiteturais das transformadas DCTs. Uma das abordagens será a redução do nível de paralelismo, para reduzir o custo de hardware. Deste modo, pretende-se gerar versões para todos os tamanhos de transformada com a mesma taxa de consumo de amostras. Inicialmente, o uso de duas e quatro amostras por ciclo será investigado. Além disso, versões com *pipeline* destas arquiteturas também serão desenvolvidas em trabalhos futuros, com o objetivo de manter uma elevada taxa de processamento e um nível de paralelismo mais reduzido em termos do número de amostras consumidas por ciclo de *clock*. Finalmente, uma versão *standard-cells* das arquiteturas desenvolvidas também está planejada para trabalhos futuros.

REFERÊNCIAS

AGOSTINI, L. **Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas a Compressão de Vídeo Segundo o Padrão H.264/AVC**. 2007. 172f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

AHMED, A. et al. **VLSI implementation of 16-point DCT for H.265/HEVC using walsh hadamard transform and lifting scheme**. *Multitopic Conference (INMIC)*, Pakistan, 2011, pp. 144-148.

BUDAGAVI, M. et al. **Unified Forward+Inverse Transform Architecture For Hevc**. *Image Processing (ICIP),* Orlando, Los Alamitos, 2012 Paper: MA.P2.8.

CHENG, W. et al. **A Novel 8×8 Transform Method Applied In Video Coding.** *Wireless Mobile and Computing (CCWMC 2011),* Shanghai, 2011, pp. 54–59.

EDIRISURIYA, A. et al. **A Multiplication-free Digital Architecture for 16x16 2-D DCT/DST Transform for HEVC.** *Electrical & Electronics Engineers in Israel (IEEEI),* Israel, 2012, pp. 1–5.

GHANBARI, M. **Standard Codecs: Image Compression to Advanced Video Coding.** United Kingdom: The Institution of Electrical Engineers, 2003.

GHISSONI, S. et al. **Combination of constant matrix multiplication and gate-level approaches for area and power efficient hybrid radix-2 DIT FFT realization.** in *IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*, 2011 18th, Beirute, Libano, 2011, pp. 567-570.

GNU (GPROF). Available at: <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html>. Acesso em 15 de novembro de 2011.

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

HAN, W., et al. **Improved video compression efficiency through Flexible Unit Representation and Corresponding Extension of Coding Tools.** IEEE Transactions on Circuits and Systems for Video Technology. Vol. 20, pp. 1709-1720. Dezembro de 2010.

INTERNATIONAL TELECOMMUNICATION UNION. ITU-T Recommendation H.262 (11/94): **generic coding of moving pictures and associated audio information – part 2: video.** 1994.

INTERNATIONAL TELECOMMUNICATION UNION. ITU-T Recommendation H.264/AVC (03/05): **advanced video coding for generic audiovisual services.** 2005.

JESKE, R. et. al. **Low cost and high throughput multiplierless design of a 16 point 1-D DCT of the new HEVC video coding standard.** *Programmable Logic (SPL)*, Bento Gonçalves, RS, 2012, pp.1-6.

JOINT COLLABORATIVE TEAM ON VIDEO CODING (JCT-VC). Available at: <http://www.itu.int/en/ITU-T/studygroups/com16/video/Pages/jctvc.aspx>. Acesso em 10 de dezembro de 2010.

JOINT COLLABORATIVE TEAM ON VIDEO CODING (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 – **HM4: High Efficiency Video Coding (HEVC) Test Model 4 Encoder Description.** 6th Meeting: Torino, IT, 14-22 Julho, 2011.

JOINT COLLABORATIVE TEAM ON VIDEO CODING (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 – **HM9: High Efficiency Video Coding (HEVC) Test Model 9 Encoder Description.** 11th Meeting: Shanghai, CN, 10–19 Outubro 2012.

JOINT COLLABORATIVE TEAM ON VIDEO CODING (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 **Proposed editorial improvements for High Efficiency Video Coding (HEVC) text specification draft 9 (SoDIS).** 12th Meeting: Geneva, CN, 14–23 Janeiro 2013.

MARTUZA, M. et al. **A cost effective implementation of 8×8 transform of HEVC from H.264/AVC.** Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on, Montreal, Quebec, pp. 1-4.

NETBEANS. Available at: < http://netbeans.org/>. Acesso em 10 de março de 2011.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression :** Video Coding for Next-Generation Multimedia. Chichester: John Wiley and Sons, 2003.

RICHARDSON, I. E-book. **The H.264 Advanced Video Compression Standard.** 2a.Ed. John Wiley & Sons, 2010.

SHEN, S. et al. **A unified 4/8/16/32-point integer idct architecture for Multiple video coding standards.** Multimedia and Expo (ICME), 2012 IEEE International Conference on, Melbourne, 9-13 July 2012, pp. 788-793.

APÊNCICE A: EQUAÇÕES GERADAS PELO SOFTWARE DE OTIMIZAÇÃO

DCT de 4 de pontos

Tabela A.1 - Segunda etapa de operações da DCT de 4-pontos (bs)

Etapa 'bs' da DCT de 4-pontos	
b1 = a1 + a2	b2 = a1 - a2

Tabela A.2 - Operações finais geradas pelo software da DCT de 4-pontos

EOS da DCT de 4-pontos	
saida2	= 64*b1 + 32*b2 - 16 a1 + 4*b1 - a1
saida4	= 64*b2 - 32*b1 + 16*a2 + 4*b2 + a2

Tabela A.3 - Total de operações geradas pelo software da DCT de 4-pontos

DCT 4-pontos	Multiplicações	Adições
1D	0	80
2D	0	160

DCT de 8 de pontos

Tabela A.4 - Segunda etapa de operações da DCT de 8-pontos (bs)

Etapa 'bs' da DCT de 8-pontos			
b1 = a1 + a2	b3 = a3 + a4	b5 = a1 - a3	b7 = a2 - a4
b2 = a1 + a3	b4 = a1 - a2	b6 = a2 + a4	b8 = a3 - a4

Tabela A.5 - Operações finais da DCT de 8-pontos geradas pelo software

EOS da DCT de 8-pontos	
saida1 =	64*b1 + 32*b2 + 16*b3 - 8*b4 + 4*a2 + 2*b3 + b4
saida2 =	64*b5 - 32*b3 - 16*b6 + 8*b2 + 4*a1 - 2*b6 - b2
saida3 =	-64*b7 + 32*b4 + 16*b2 + 8*b6 + 4*a4 + 2*b2 - b6
saida4 =	64*b8 - 32*b6 + 16*b4 + 8*b3 + 4*a3 + 2*b4 - b3

Tabela A.6 - Total de operações da DCT de 8-pontos

DCT 8-pontos	Multiplicações	Adições
1D	0	512
2D	0	1024

DCT de 16 de pontos

Tabela A.7 - Segunda etapa de operações da DCT de 16-pontos (bs)

'bs'			
b1 = a1 - a4	b4 = a1 - a5	b7 = a2 + a3	b10 = a1 + a4
b2 = a6 - a7	b5 = a4 - a8	b8 = a5 + a8	b11 = a5 - a8
b3 = a3 + a7	b6 = a2 + a6	b9 = a6 + a7	b12 = a2 - a3

Tabela A.8 - Terceira etapa de operações da DCT de 16-pontos (cs)

'cs'			
c1 = a1 + b6	c7 = a6 - b4	c13 = a3 - b5	c19 = a3 - b9
c2 = a5 - b3	c8 = a8 - b3	c14 = a3 + b8	c20 = a6 - b12
c3 = a6 - b1	c9 = a5 - b7	c15 = a7 + b1	c21 = a5 + b10
c4 = a2 + b8	c10 = a8 + b2	c16 = a4 - b6	c22 = a1 + b11
c5 = a4 - b2	c11 = a2 - b4	c17 = a2 - b9	c23 = a4 - b11
c6 = a7 - b5	c12 = a1 + b7	c18 = a8 + b10	c24 = a7 + b12

Tabela A.9 - Quarta etapa de operações da DCT de 16-pontos (ds)

'ds'			
d1 = b7 + c21	d5 = b1 - c19	d9 = b6 - c18	d13 = b5 + c24
d2 = b8 - c17	d6 = b6 + c23	d10 = b4 - c19	d14 = b2 + c22
d3 = b4 + c17	d7 = b3 - c22	d11 = b8 - c20	d15 = b2 + c23
d4 = b7 - c18	d8 = b5 + c20	d12 = b3 + c21	d16 = b1 - c24

Tabela A.10 - Saídas das 8 equações específicas da DCT de 16-pontos

8 saídas específicas da DCT de 16-pontos	
saida1 =	64*d1 + 32*c1 - 16*c2 + 8*d2 - 4*b1 + 2*c3 + d2
saida2 =	64*d3 - 32*c3 - 16*c4 + 8*d4 + 4*b2 - 2*c5 + d4
saida3 =	64*d5 + 32*c6 - 16*c7 + 8*d6 - 4*b3 + 2*c8 + d6
saida4 =	-64*d7 + 32*c9 + 16*c10 + 8*d8 + 4*b4 - 2*c11 + d8
saida5 =	-64*d9 + 32*c5 - 16*c12 + 8*d10 - 4*b5 + 2*c6 + d10
saida6 =	-64*d11 - 32*c11 - 16*c13 + 8*d12 + 4*b6 + 2*c1 + d12
saida7 =	-64*d13 + 32*c14 + 16*c15 + 8*d14 - 4*b7 + 2*c9 + d14
saida8 =	-64*d15 - 32*c8 + 16*c16 + 8*d16 + 4*b8 + 2*c14 + d16

Tabela A.11 - Saídas das 8 equações específicas da DCT de 16-pontos

DCT 16x16	Multiplicações	Adições
1D	0	3008
2D	0	6016

DCT de 32 de pontos

Tabela A.12 - Segunda etapa de operações da DCT de 32-pontos (bs)

'bs'			
b1 = a1 - a12	b15 = a9 - a12	b29 = a1 + a8	b43 = a3 - a4
b2 = a6 + a10	b16 = a11 - a15	b30 = a8 + a16	b44 = a8 - a9
b3 = a7 + a11	b17 = a4 + a5	b31 = a2 - a7	b45 = a13 + a14
b4 = a5 + a16	b18 = a4 + a12	b32 = a2 - a10	b46 = a8 + a9
b5 = a14 - a15	b19 = a6 - a14	b33 = a6 + a12	b47 = a10 - a12
b6 = a2 + a3	b20 = a11 + a14	b34 = a1 - a16	b48 = a3 + a16
b7 = a4 - a9	b21 = a3 - a11	b35 = a13 + a15	b49 = a2 + a15
b8 = a8 + a13	b22 = a5 + a13	b36 = a8 - a11	b50 = a1 - a10
b9 = a1 + a13	b23 = a12 - a13	b37 = a7 - a10	b51 = a2 + a9
b10 = a2 - a6	b24 = a3 - a6	b38 = a8 - a15	b52 = a6 + a9
b11 = a3 - a7	b25 = a9 - a16	b39 = a7 + a16	b53 = a7 + a10
b12 = a10 - a14	b26 = a10 + a15	b40 = a1 - a14	b54 = a2 + a4
b13 = a4 + a16	b27 = a7 - a15	b41 = a2 - a15	b55 = a5 + a11
b14 = a5 + a8	b28 = a1 + a9	b42 = a5 - a7	b56 = a1 + a16

Tabela A.13 - Terceira etapa de operações da DCT de 32-pontos (cs)

'cs'			
c1 = b2 + b3	c26 = a14 + b13	c51 = b2 - b13	c76 = a11 + b5
c2 = b9 - b13	c27 = b10 + b16	c52 = b23 + b29	c77 = b7 - b10
c3 = b1 - b4	c28 = b5 - b6	c53 = b7 - b12	c78 = a4 + b4
c4 = b3 + b7	c29 = b7 + b8	c54 = b22 - b29	c79 = b2 + b9
c5 = a3 + b9	c30 = b4 + b5	c55 = b5 - b15	c80 = b5 - b8
c6 = b24 - b32	c31 = a11 - b15	c56 = b22 - b30	c81 = b1 + b5
c7 = a12 - b12	c32 = b24 - b31	c57 = a16 + b7	c82 = b3 + b8
c8 = b19 + b32	c33 = a4 + b10	c58 = b2 - b16	c83 = b1 + b2
c9 = b11 + b15	c34 = b5 + b6	c59 = a8 + b4	c84 = a12 + b7
c10 = b19 - b26	c35 = b11 + b12	c60 = b6 - b12	c85 = b5 - b14
c11 = b9 - b14	c36 = b2 - b3	c61 = a7 - b6	c86 = a14 - b2
c12 = b10 - b11	c37 = b14 - b15	c62 = b8 + b15	c87 = b4 - b11
c13 = b1 + b6	c38 = b13 + b16	c63 = a15 - b3	c88 = a3 - b3
c14 = a13 - b16	c39 = b16 - b30	c64 = b1 - b13	c89 = b10 - b16
c15 = a6 - b14	c40 = b1 - b17	c65 = a9 + b1	c90 = b1 - b12
c16 = b7 - b8	c41 = b1 - b22	c66 = b5 - b11	c91 = a5 + b8
c17 = b20 + b26	c42 = b8 + b16	c67 = a10 - b5	c92 = b6 - b15
c18 = b10 - b15	c43 = b17 + b25	c68 = b7 + b14	c93 = b3 - b4
c19 = b20 - b27	c44 = b3 - b9	c69 = b9 + b13	c94 = b2 + b7
c20 = b2 - b8	c45 = b17 - b30	c70 = a1 + b8	c95 = b4 - b6
c21 = a5 + b11	c46 = b18 + b25	c71 = b3 - b10	c96 = b6 + b7
c22 = b1 + b4	c47 = b6 + b14	c72 = a2 - b2	c97 = a13 + b1
c23 = b21 + b27	c48 = b18 + b28	c73 = b4 + b9	c98 = b3 - b13
c24 = b12 + b13	c49 = b4 + b12	c74 = b14 + b15	c99 = a6 - b6
c25 = b21 - b31	c50 = b23 + b28	c75 = b11 - b12	

Tabela A.14 - Quarta etapa de operações da DCT de 32-pontos (ds)

‘ds’			
d1 = c13 + c14	d25 = b19 - c74	d49 = b19 - c92	d73 = b54 + c74
d2 = b17 + c76	d26 = c30 + c31	d50 = b10 - c39	d74 = b24 - c98
d3 = b33 - c35	d27 = b24 + c59	d51 = c13 - c15	d75 = b29 + c35
d4 = b34 - c77	d28 = b40 - c29	d52 = b20 - c65	d76 = c30 + c33
d5 = b27 + c29	d29 = b41 - c83	d53 = b48 - c16	d77 = b23 + c99
d6 = c63 + c64	d30 = b17 - c1	d54 = b49 - c93	d78 = b55 + c75
d7 = b32 + c78	d31 = c61 - c62	d55 = b23 - c36	d79 = b56 + c42
d8 = b35 - c37	d32 = b26 - c84	d56 = c70 - c71	d80 = b32 - c16
d9 = b20 - c79	d33 = b42 + c2	d57 = b25 + c63	d81 = c11 + c18
d10 = b25 - c75	d34 = b21 - c85	d58 = b50 - c28	d82 = c35 + c39
d11 = c4 - c5	d35 = b28 - c27	d59 = b23 + c94	d83 = c1 + c47
d12 = b19 - c57	d36 = c4 + c7	d60 = b21 - c37	d84 = c28 - c44
d13 = b36 + c3	d37 = b22 + c86	d61 = c65 - c66	d85 = c2 - c9
d14 = b37 + c80	d38 = b43 - c27	d62 = b30 - c67	d86 = c3 + c4
d15 = b18 + c34	d39 = b44 + c87	d63 = b51 + c36	d87 = c11 + c24
d16 = c59 + c60	d40 = b26 - c22	d64 = b18 + c95	d88 = c12 + c49
d17 = b28 - c61	d41 = c20 - c21	d65 = b20 + c69	d89 = c20 + c22
d18 = b38 - c1	d42 = b18 - c88	d66 = c20 - c26	d90 = c12 - c42
d19 = b22 + c81	d43 = b45 + c89	d67 = b21 + c70	d91 = c13 + c16
d20 = b24 + c2	d44 = b46 + c90	d68 = b52 - c22	d92 = c27 - c53
d21 = c57 - c58	d45 = b31 + c3	d69 = b53 - c96	d93 = c29 - c30
d22 = b29 + c72	d46 = c67 - c68	d70 = b22 - c28	d94 = c34 + c51
d23 = b39 + c34	d47 = b31 + c91	d71 = c72 + c73	d95 = c36 + c55
d24 = b17 + c82	d48 = b47 - c69	d72 = b27 - c97	d96 = c37 - c38

Tabela A.15 - Quinta etapa de operações da DCT de 32-pontos (es)

‘es’			
e1 = c48 + d83	e9 = c40 - d82	e17 = c52 - d94	e25 = c46 + d90
e2 = c17 + d81	e10 = c6 + d86	e18 = c23 + d87	e26 = c10 + d91
e3 = c17 - d91	e11 = c10 + d85	e19 = c8 - d86	e27 = c6 + d96
e4 = c45 + d84	e12 = c41 - d82	e20 = c54 - d95	e28 = c52 + d92
e5 = c19 + d81	e13 = c23 - d89	e21 = c25 + d87	e29 = c32 - d93
e6 = c43 + d90	e14 = c46 + d83	e22 = c48 - d88	e30 = c50 - d94
e7 = c54 - d92	e15 = c43 + d84	e23 = c50 - d88	e31 = c56 - d95
e8 = c25 - d93	e16 = c8 + d85	e24 = c19 + d89	e32 = c32 + d96

Tabela A.16 - 16 equações finais da DCT de 32-pontos

16 saídas específicas da DCT de 32-pontos		
saida1	=	64*e1 + 32*d1 + 16*d2 + 8*d3 - 4*d4 - 2*e2 + d5
saida2	=	-64*e3 + 32*d6 + 16*d7 - 8*d8 + 4*d9 - 2*e4 - d10
saida3	=	64*e5 - 32*d11 - 16*d12 - 8*d13 + 4*d14 + 2*e6 + d15
saida4	=	-64*e7 - 32*d16 + 16*d17 + 8*d18 + 4*d19 + 2*e8 + d20
saida5	=	64*e9 - 32*d21 + 16*d22 + 8*d23 + 4*d24 + 2*e10 - d25
saida6	=	64*e11 + 32*d26 - 16*d27 + 8*d28 - 4*d29 + 2*e12 - d30
saida7	=	-64*e13 + 32*d31 + 16*d32 + 8*d33 + 4*d34 + 2*e14 + d35
saida8	=	64*e15 - 32*d36 + 16*d37 - 8*d38 + 4*d39 + 2*e16 - d40
saida9	=	64*e17 - 32*d41 + 16*d42 - 8*d43 - 4*d44 + 2*e18 - d45
saida10	=	-64*e19 - 32*d46 - 16*d47 + 8*d48 + 4*d49 + 2*e20 - d50
saida11	=	64*e21 - 32*d51 - 16*d52 + 8*d53 + 4*d54 - 2*e22 - d55
saida12	=	64*e23 - 32*d56 + 16*d57 + 8*d58 - 4*d59 - 2*e24 + d60
saida13	=	64*e25 + 32*d61 + 16*d62 - 8*d63 + 4*d64 - 2*e26 + d65
saida14	=	64*e27 + 32*d66 + 16*d67 - 8*d68 - 4*d69 - 2*e28 + d70
saida15	=	64*e29 + 32*d71 + 16*d72 - 8*d73 - 4*d74 - 2*e30 - d75
saida16	=	64*e31 - 32*d76 - 16*d77 + 8*d78 + 4*d79 - 2*e32 + d80

Tabela A.17 - Total de operações DCT de 32-pontos

DCT 32x32	Multiplicações	Adições
1D	0	19.680
2D	0	39.360

APÊNDICE B: PUBLICAÇÕES GERADAS A PARTIR DESTE TRABALHO

Os resultados parciais do trabalho apresentado nesta dissertação geraram as seguintes publicações:

- (JESKE, de SOUZA JR., WREGE, CONCEIÇÃO, GRELLERT, MATTOS, AGOSTINI) - **Low cost and high throughput multiplierless design of a 16 point 1-D DCT of the new HEVC video coding standard.** *Programmable Logic (SPL)*, Bento Gonçalves, RS, 2012, pp.1-6.
- (CONCEIÇÃO, JESKE, AGOSTINI, MATTOS) - **Software Solution for Hardware Optimization of 1-D DCT of the HEVC**. *SForum 2012*, 2012, Brasília, 2012.
- (JESKE, de SOUZA JR., CONCEIÇÃO, MATTOS, AGOSTINI) **Projeto em Hardware com Baixo Custo e Elevada Taxa de Processamento para a Transformada DCT 16x16 do Padrão Emergente de Codificação de Vídeos HEVC**. XIV encontro de Pós-Graduação da UFPel (ENPOS, 2012). Pelotas, 2012.
  (Menção Honrosa na área de Ciências Exatas e da Terra, na modalidade Apresentação Oral)
- (HECKTHEUER, CONCEIÇÃO, WREGE, de SOUZA JR., JESKE, AGOSTINI, MATTOS) - **Reconfigurable Architecture for 1-D Discrete Cosine Transform of the HEVC Emerging Video Encoding Standard.** XXVII Simpósio Sul de Microeletrônica, 2012, (SIM 2012), São Miguel das Missões, 2012.
- (JESKE, de SOUZA JR., WREGE, CONCEIÇÃO, MATTOS, AGOSTINI) - **Projeto de Baixo Custo e Elevada Taxa de Processamento Sem Multiplicadores da Transformada DCT 1-D de 16 Pontos para o Novo Padrão HEVC de Codificação de Vídeo.** II Seminário de Pesquisa em Computação da UFPel, (SPC 2011), Pelotas, 2011.

ANEXO A: MATRIZES DE CONSTANTES DAS TRANSFORMADAS

Este anexo apresenta as matrizes das transformadas c_{ij} (i, j=0..n-1) para n = 4, 8, 16, e 32. Conforme especificadas no HM9.

Matriz de Constantes DCT 1D Direta 4x4

Tabela An.7.18 – Constantes DCT 4x4

n = 4 (g_aiT4[4][4])			
64	64	64	64
83	36	-36	-83
64	-64	-64	64
36	-83	83	-36

Matriz de Constantes DCT 1D Direta 8x8

Tabela An.7.19 – Constantes DCT 8x8

n = 8 (g_aiT8[8][8])							
64	64	64	64	64	64	64	64
89	75	50	18	-18	-50	-75	-89
83	36	-36	-83	-83	-36	36	83
75	-18	-89	-50	50	89	18	-75
64	-64	-64	64	64	-64	-64	64
50	-89	18	75	-75	-18	89	-50
36	-83	83	-36	-36	83	-83	36
18	-50	75	-89	89	-75	50	-18

Matriz de Constantes DCT 1D Direta 16x16

Tabela An.7.20 – Constantes DCT 16x16

n = 16 (g_aiT16[16][16])															
64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
90	87	80	70	57	43	25	9	-9	-25	-43	-57	-70	-80	-87	-90
89	75	50	18	-18	-50	-75	-89	-89	-75	-50	-18	18	50	75	89
87	57	9	-43	-80	-90	-70	-25	25	70	90	80	43	-9	-57	-87
83	36	-36	-83	-83	-36	36	83	83	36	-36	-83	-83	-36	36	83
80	9	-70	-87	-25	57	90	43	-43	-90	-57	25	87	70	-9	-80
75	-18	-89	-50	50	89	18	-75	-75	18	89	50	-50	-89	-18	75
70	-43	-87	9	90	25	-80	-57	57	80	-25	-90	-9	87	43	-70
64	-64	-64	64	64	-64	-64	64	64	-64	-64	64	64	-64	-64	64
57	-80	-25	90	-9	-87	43	70	-70	-43	87	9	-90	25	80	-57
50	-89	18	75	-75	-18	89	-50	-50	89	-18	-75	75	18	-89	50
43	-90	57	25	-87	70	9	-80	80	-9	-70	87	-25	-57	90	-43
36	-83	83	-36	-36	83	-83	36	36	-83	83	-36	-36	83	-83	36
25	-70	90	-80	43	9	-57	87	-87	57	-9	-43	80	-90	70	-25
18	-50	75	-89	89	-75	50	-18	-18	50	-75	89	-89	75	-50	18
9	-25	43	-57	70	-80	87	-90	90	-87	80	-70	57	-43	25	-9

Matriz de Constantes DCT 1D Direta 32x32

Tabela An.7.21 - Constantes DCT 32x32

n = 32 (g_aiT32[32][32])																															
64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
90	90	88	85	82	78	73	67	61	54	46	38	31	22	13	4	-4	-13	-22	-31	-38	-46	-54	-61	-67	-73	-78	-82	-85	-88	-90	-90
90	87	80	70	57	43	25	9	-9	-25	-43	-57	-70	-80	-87	-90	-90	-87	-80	-70	-57	-43	-25	-9	9	25	43	57	70	80	87	90
90	82	67	46	22	-4	-31	-54	-73	-85	-90	-88	-78	-61	-38	-13	13	38	61	78	88	90	85	73	54	31	4	-22	-46	-67	-82	-90
89	75	50	18	-18	-50	-75	-89	-89	-75	-50	-18	18	50	75	89	89	75	50	18	-18	-50	-75	-89	-89	-75	-50	-18	18	50	75	89
88	67	31	-13	-54	-82	-90	-78	-46	-4	38	73	90	85	61	22	-22	-61	-85	-90	-73	-38	4	46	78	90	82	54	13	-31	-67	-88
87	57	9	-43	-80	-90	-70	-25	25	70	90	80	43	-9	-57	-87	-87	-57	-9	43	80	90	70	25	-25	-70	-90	-80	-43	9	57	87
85	46	-13	-67	-90	-73	-22	38	82	88	54	-4	-61	-90	-78	-31	31	78	90	61	4	-54	-88	-82	-38	22	73	90	67	13	-46	-85
83	36	-36	-83	-83	-36	36	83	83	36	-36	-83	-83	-36	36	83	83	36	-36	-83	-83	-36	36	83	83	36	-36	-83	-83	-36	36	83
82	22	-54	-90	-61	13	78	85	31	-46	-90	-67	4	73	88	38	-38	-88	-73	-4	67	90	46	-31	-85	-78	-13	61	90	54	-22	-82
80	9	-70	-87	-25	57	90	43	-43	-90	-57	25	87	70	-9	-80	-80	-9	70	87	25	-57	-90	-43	43	90	57	-25	-87	-70	9	80
78	-4	-82	-73	13	85	67	-22	-88	-61	31	90	54	-38	-90	-46	46	90	38	-54	-90	-31	61	88	22	-67	-85	-13	73	82	4	-78
75	-18	-89	-50	50	89	18	-75	-75	18	89	50	-50	-89	-18	75	75	-18	-89	-50	50	89	18	-75	-75	18	89	50	-50	-89	-18	75
73	-31	-90	-22	78	67	-38	-90	-13	82	61	-46	-88	-4	85	54	-54	-85	4	88	46	-61	-82	13	90	38	-67	-78	22	90	31	-73
70	-43	-87	9	90	25	-80	-57	57	80	-25	-90	-9	87	43	-70	-70	43	87	-9	-90	-25	80	57	-57	-80	25	90	9	-87	-43	70
67	-54	-78	38	85	-22	-90	4	90	13	-88	-31	82	46	-73	-61	61	73	-46	-82	31	88	-13	-90	-4	90	22	-85	-38	78	54	-67
64	-64	-64	64	64	-64	-64	64	64	-64	-64	64	64	-64	-64	64	64	-64	-64	64	64	-64	-64	64	64	-64	-64	64	64	-64	-64	64
61	-73	-46	82	31	-88	-13	90	-4	-90	22	85	-38	-78	54	67	-67	-54	78	38	-85	-22	90	4	-90	13	88	-31	-82	46	73	-61
57	-80	-25	90	-9	-87	43	70	-70	-43	87	9	-90	25	80	-57	-57	80	25	-90	9	87	-43	-70	70	43	-87	-9	90	-25	-80	57
54	-85	-4	88	-46	-61	82	13	-90	38	67	-78	-22	90	-31	-73	73	31	-90	22	78	-67	-38	90	-13	-82	61	46	-88	4	85	-54
50	-89	18	75	-75	-18	89	-50	-50	89	-18	-75	75	18	-89	50	50	-89	18	75	-75	-18	89	-50	-50	89	-18	-75	75	18	-89	50
46	-90	38	54	-90	31	61	-88	22	67	-85	13	73	-82	4	78	-78	-4	82	-73	-13	85	-67	-22	88	-61	-31	90	-54	-38	90	-46
43	-90	57	25	-87	70	9	-80	80	-9	-70	87	-25	-57	90	-43	-43	90	-57	-25	87	-70	-9	80	-80	9	70	-87	25	57	-90	43
38	-88	73	-4	-67	90	-46	-31	85	-78	13	61	-90	54	22	-82	82	-22	-54	90	-61	-13	78	-85	31	46	-90	67	4	-73	88	-38
36	-83	83	-36	-36	83	-83	36	36	-83	83	-36	-36	83	-83	36	36	-83	83	-36	-36	83	-83	36	36	-83	83	-36	-36	83	-83	36
31	-78	90	-61	4	54	-88	82	-38	-22	73	-90	67	-13	-46	85	-85	46	13	-67	90	-73	22	38	-82	88	-54	-4	61	-90	78	-31
25	-70	90	-80	43	9	-57	87	-87	57	-9	-43	80	-90	70	-25	-25	70	-90	80	-43	-9	57	-87	87	-57	9	43	-80	90	-70	25
22	-61	85	-90	73	-38	-4	46	-78	90	-82	54	-13	-31	67	-88	88	-67	31	13	-54	82	-90	78	-46	4	38	-73	90	-85	61	-22
18	-50	75	-89	89	-75	50	-18	-18	50	-75	89	-89	75	-50	18	18	-50	75	-89	89	-75	50	-18	-18	50	-75	89	-89	75	-50	18
13	-38	61	-78	88	-90	85	-73	54	-31	4	22	-46	67	-82	90	-90	82	-67	46	-22	-4	31	-54	73	-85	90	-88	78	-61	38	-13
9	-25	43	-57	70	-80	87	-90	90	-87	80	-70	57	-43	25	-9	-9	25	-43	57	-70	80	-87	90	-90	87	-80	70	-57	43	-25	9
4	-13	22	-31	38	-46	54	-61	67	-73	78	-82	85	-88	90	-90	90	-90	88	-85	82	-78	73	-67	61	-54	46	-38	31	-22	13	-4

ANEXO B: ALGORITMOS DAS TRANSFORMADAS DCT

Transformada DCT 1D Direta 4x4

```
void partialButterfly4(short src[4][4],short dst[4][4],int shift)
{
  int j;
  int E[2],O[2];
  int add = 1<<(shift-1);

  for (j=0; j<4; j++)
  {
    /* E and O */
    E[0] = src[j][0] + src[j][3];
    O[0] = src[j][0] - src[j][3];
    E[1] = src[j][1] + src[j][2];
    O[1] = src[j][1] - src[j][2];

    dst[0][j] = (g_aiT4[0][0]*E[0] + g_aiT4[0][1]*E[1] + add)>>shift;
    dst[2][j] = (g_aiT4[2][0]*E[0] + g_aiT4[2][1]*E[1] + add)>>shift;
    dst[1][j] = (g_aiT4[1][0]*O[0] + g_aiT4[1][1]*O[1] + add)>>shift;
    dst[3][j] = (g_aiT4[3][0]*O[0] + g_aiT4[3][1]*O[1] + add)>>shift;
  }
}
```

Transformada DCT 1D Direta 8x8

```
void partialButterfly8(short src[8][8],short dst[8][8],int shift)
{
  int j,k;
  int E[4],O[4];
  int EE[2],EO[2];
  int add = 1<<(shift-1);

  for (j=0; j<8; j++)
  {
    /* E and O*/
    for (k=0;k<4;k++)
    {
      E[k] = src[j][k] + src[j][7-k];
      O[k] = src[j][k] - src[j][7-k];
    }
    /* EE and EO */
    EE[0] = E[0] + E[3];
    EO[0] = E[0] - E[3];
    EE[1] = E[1] + E[2];
    EO[1] = E[1] - E[2];

    dst[0][j] = (g_aiT8[0][0]*EE[0] + g_aiT8[0][1]*EE[1] + add)>>shift;
    dst[4][j] = (g_aiT8[4][0]*EE[0] + g_aiT8[4][1]*EE[1] + add)>>shift;
    dst[2][j] = (g_aiT8[2][0]*EO[0] + g_aiT8[2][1]*EO[1] + add)>>shift;
    dst[6][j] = (g_aiT8[6][0]*EO[0] + g_aiT8[6][1]*EO[1] + add)>>shift;

    dst[1][j] = (g_aiT8[1][0]*O[0] + g_aiT8[1][1]*O[1] + g_aiT8[1][2]*O[2] + g_aiT8[1][3]*O[3] +
add)>>shift;
    dst[3][j] = (g_aiT8[3][0]*O[0] + g_aiT8[3][1]*O[1] + g_aiT8[3][2]*O[2] + g_aiT8[3][3]*O[3] +
add)>>shift;
    dst[5][j] = (g_aiT8[5][0]*O[0] + g_aiT8[5][1]*O[1] + g_aiT8[5][2]*O[2] + g_aiT8[5][3]*O[3] +
add)>>shift;
```

```
   dst[7][j] = (g_aiT8[7][0]*O[0] + g_aiT8[7][1]*O[1] + g_aiT8[7][2]*O[2] + g_aiT8[7][3]*O[3] +
add)>>shift;
  }
}
```

Transformada DCT 1D Direta 16x16

```
void partialButterfly16(short src[16][16],short dst[16][16],int shift)
{
  int j,k;
  int E[8],O[8];
  int EE[4],EO[4];
  int EEE[2],EEO[2];
  int add = 1<<(shift-1);

  for (j=0; j<16; j++)
  {
    /* E and O*/
    for (k=0;k<8;k++)
    {
      E[k] = src[j][k] + src[j][15-k];
      O[k] = src[j][k] - src[j][15-k];
    }
    /* EE and EO */
    for (k=0;k<4;k++)
    {
      EE[k] = E[k] + E[7-k];
      EO[k] = E[k] - E[7-k];
    }
    /* EEE and EEO */
    EEE[0] = EE[0] + EE[3];
    EEO[0] = EE[0] - EE[3];
    EEE[1] = EE[1] + EE[2];
    EEO[1] = EE[1] - EE[2];

    dst[ 0][j] = (g_aiT16[ 0][0]*EEE[0] + g_aiT16[ 0][1]*EEE[1] + add)>>shift;
    dst[ 8][j] = (g_aiT16[ 8][0]*EEE[0] + g_aiT16[ 8][1]*EEE[1] + add)>>shift;
    dst[ 4][j] = (g_aiT16[ 4][0]*EEO[0] + g_aiT16[ 4][1]*EEO[1] + add)>>shift;
    dst[12][j] = (g_aiT16[12][0]*EEO[0] + g_aiT16[12][1]*EEO[1] + add)>>shift;

    for (k=2;k<16;k+=4)
    {
      dst[k][j] = (g_aiT16[k][0]*EO[0] + g_aiT16[k][1]*EO[1] + g_aiT16[k][2]*EO[2] +
g_aiT16[k][3]*EO[3] + add)>>shift;
    }

    for (k=1;k<16;k+=2)
    {
      dst[k][j] = (g_aiT16[k][0]*O[0] + g_aiT16[k][1]*O[1] + g_aiT16[k][2]*O[2] + g_aiT16[k][3]*O[3]
+
                   g_aiT16[k][4]*O[4] + g_aiT16[k][5]*O[5] + g_aiT16[k][6]*O[6] + g_aiT16[k][7]*O[7] +
add)>>shift;
    }
  }
}
```

Transformada DCT 1D Direta 32x32

```
void partialButterfly32(short src[32][32],short dst[32][32],int shift)
{
  int j,k;
  int E[16],O[16];
```

```
  int EE[8],EO[8];
  int EEE[4],EEO[4];
  int EEEE[2],EEEO[2];
  int add = 1<<(shift-1);

  for (j=0; j<32; j++)
  {
    /* E and O*/
    for (k=0;k<16;k++)
    {
      E[k] = src[j][k] + src[j][31-k];
      O[k] = src[j][k] - src[j][31-k];
    }
    /* EE and EO */
    for (k=0;k<8;k++)
    {
      EE[k] = E[k] + E[15-k];
      EO[k] = E[k] - E[15-k];
    }
    /* EEE and EEO */
    for (k=0;k<4;k++)
    {
      EEE[k] = EE[k] + EE[7-k];
      EEO[k] = EE[k] - EE[7-k];
    }
    /* EEEE and EEEO */
    EEEE[0] = EEE[0] + EEE[3];
    EEEO[0] = EEE[0] - EEE[3];
    EEEE[1] = EEE[1] + EEE[2];
    EEEO[1] = EEE[1] - EEE[2];

    dst[ 0][j] = (g_aiT32[ 0][0]*EEEE[0] + g_aiT32[ 0][1]*EEEE[1] + add)>>shift;
    dst[16][j] = (g_aiT32[16][0]*EEEE[0] + g_aiT32[16][1]*EEEE[1] + add)>>shift;
    dst[ 8][j] = (g_aiT32[ 8][0]*EEEO[0] + g_aiT32[ 8][1]*EEEO[1] + add)>>shift;
    dst[24][j] = (g_aiT32[24][0]*EEEO[0] + g_aiT32[24][1]*EEEO[1] + add)>>shift;
    for (k=4;k<32;k+=8)
    {
      dst[k][j] = (g_aiT32[k][0]*EEO[0] + g_aiT32[k][1]*EEO[1] + g_aiT32[k][2]*EEO[2] +
g_aiT32[k][3]*EEO[3] + add)>>shift;
    }
    for (k=2;k<32;k+=4)
    {
      dst[k][j] = (g_aiT32[k][0]*EO[0] + g_aiT32[k][1]*EO[1] + g_aiT32[k][2]*EO[2] +
g_aiT32[k][3]*EO[3] +
                   g_aiT32[k][4]*EO[4] + g_aiT32[k][5]*EO[5] + g_aiT32[k][6]*EO[6] +
g_aiT32[k][7]*EO[7] + add)>>shift;
    }
    for (k=1;k<32;k+=2)
    {
      dst[k][j] = (g_aiT32[k][ 0]*O[ 0] + g_aiT32[k][ 1]*O[ 1] + g_aiT32[k][ 2]*O[ 2] + g_aiT32[k][
3]*O[ 3] +
                   g_aiT32[k][ 4]*O[ 4] + g_aiT32[k][ 5]*O[ 5] + g_aiT32[k][ 6]*O[ 6] + g_aiT32[k][
7]*O[ 7] +
                   g_aiT32[k][ 8]*O[ 8] + g_aiT32[k][ 9]*O[ 9] + g_aiT32[k][10]*O[10] +
g_aiT32[k][11]*O[11] +
                   g_aiT32[k][12]*O[12] + g_aiT32[k][13]*O[13] + g_aiT32[k][14]*O[14] +
g_aiT32[k][15]*O[15] + add)>>shift;
    }
  }
}
```