UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

UNIDADE ACADÊMICA DO CABO DE SANTO AGOSTINHO

BACHARELADO EM ENGENHARIA ELETRÔNICA

LEONARDO NOGUEIRA LINDOLFO DA SILVA

Implementação em VHDL e análise comparativa da Transformada Discreta do Cosseno e a aproximação de Lengwehasatit-Ortega

Cabo de Santo Agostinho – PE
2024

LEONARDO NOGUEIRA LINDOLFO DA SILVA

Implementação em VHDL e análise comparativa da Transformada Discreta do Cosseno e a aproximação de Lengwehasatit-Ortega

Monografia apresentada à Universidade Federal Rural de Pernambuco como requisito para a obtenção do título de bacharel em engenharia eletrônica

Área de concentração: Processamento Digital de Sinais

Orientador: Prof. Dr. Felipe Alberto Barbosa Simão Ferreira

Cabo de Santo Agostinho – PE
2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

L581i da Silva, Leonardo Nogueira Lindolfo
Implementação em VHDL e análise comparativa da Transformada Discreta do Cosseno e a aproximação de Lengwehasatit-Ortega / Leonardo Nogueira Lindolfo da Silva. - 2024.
48 f. : il.

Orientador: Felipe Alberto Barbosa Simao Ferreira.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, , Cabo de Santo Agostinho, 2024.

1. DCT. 2. Aproximação DCT. 3. VHDL. 4. FPGA. 5. Compressão de Imagens. I. Ferreira, Felipe Alberto Barbosa Simao, orient. II. Título

CDD

LEONARDO NOGUEIRA LINDOLFO DA SILVA

Implementação em VHDL e análise comparativa da Transformada Discreta do Cosseno e a aproximação de Lengwehasatit-Ortega

Monografia apresentada ao Curso de Graduação em Engenharia Eletrônica da Unidade Acadêmica do Cabo de Santo Agostinho da Universidade Federal Rural de Pernambuco para a obtenção do título de Bacharel em Engenharia Eletrônica.

Aprovada em: ___/___/_____

Banca Examinadora

Prof. Dr. Felipe Alberto Barbosa Simão Ferreira, UACSA, UFRPE

Orientador

Prof. Dr. Roberto Kenji Hiramatsu, UACSA, UFRPE

Examinador

Prof. Dr. Elias Marques Ferreira de Oliveira, UACSA, UFRPE

Examinador

DEDICATÓRIA

Dedico esse trabalho a minha família e amigos, que me deram suporte durante as dificuldades da vida.

AGRADECIMENTOS

À minha mãe, Nilda, que sempre me deu suporte desde o dia em que passei no vestibular até os momentos mais difíceis, sempre com palavras de sabedoria e incentivo.

Ao meu pai, João, por todo seu esforço para minha criação, trabalhando de domingo a domingo e feriados e ainda assim nunca deixou de se fazer presente.

Ao meu irmão, Lucas, pela parceria de uma vida e por sempre ter me incentivado a aproveitar os momentos e ter ajudado no contato com nossos pais auxiliando no manuseio das ferramentas de tecnologia.

Ao meu grande amigo, Reylan, que compartilhou das dores e alegrias da graduação desde as noites em claro até aos momentos de descontração e que no momento em que mais precisei esteve ao meu lado.

Ao meu orientador, professor Felipe, por ter me aceito como aluno de iniciação científica propondo um tema alinhado com linhas de pesquisa da pós-graduação de meus sonhos quando eu não mais acreditava que seria possível e desde então colaborando imensamente com minha formação pessoal e acadêmica.

“Se eu vi mais longe, foi porque estava sobre os ombros de gigantes.”

(Sir Isaac Newton, 1675)

RESUMO

O alto volume de imagens transmitidas diariamente pela internet consome grande parte da largura de banda e da capacidade de processamento. Para otimizar a compressão e manter a qualidade, são utilizadas ferramentas de processamento digital de sinais como a Transformada Discreta do Cosseno (DCT do inglês, *Discrete Cosine Transform*). A DCT é uma operação matemática que concentra a maior parte da energia do sinal em baixas frequências, sendo muito utilizada em algoritmos de compressão de imagens. O cálculo da Transformada Discreta do Cosseno é realizado utilizando multiplicação de matrizes, onde os elementos da matriz de transformação são números de ponto flutuante. Para simplificar esses cálculos, são encontradas na literatura diversas aproximações para a Transformada Discreta do Cosseno que utilizam o máximo possível de simplificações em suas matrizes de transformação. Devido à recorrência do cálculo de matrizes nos sistemas computacionais modernos especialmente para processamento de imagens e inteligência artificial, diversos sistemas apresentam ASICs (*Application Specific Integrated Circuit*) ou parte de SoCs (*System on Chip*) dedicados a essa tarefa. Nesse trabalho, foram avaliadas algumas das aproximações da DCT no contexto de compressão de imagens. A aproximação de Lengwehasatit-Ortega que apresentou o melhor desempenho, além da DCT exata foram implementadas em VHDL e sintetizadas em FPGA. Foi possível observar que seguindo a mesma filosofia de *design* a aproximação consumiu uma quantidade muito menor de recursos de *hardware* assim como era esperado.

Palavras-chave: DCT; Aproximação DCT; VHDL; FPGA; Processamento Digital de Sinais; Compressão de Imagens.

ABSTRACT

The high volume of images transmitted daily over the internet consumes a significant portion of bandwidth and processing capacity. To optimize compression and maintain quality, digital signal processing tools like the Discrete Cosine Transform (DCT) are utilized. The DCT is a mathematical operation that concentrates most of the signal's energy in low frequencies, making it highly useful in image compression algorithms. The calculation of the Discrete Cosine Transform is performed using matrix multiplication, where the elements of the transformation matrix are floating-point numbers. To simplify these calculations, various approximations for the Discrete Cosine Transform that use as many integers as possible in their transformation matrices can be found in the literature. Due to the recurrent matrix calculations in modern computational systems, especially for image processing and artificial intelligence, many systems feature ASICs (Application Specific Integrated Circuits) or parts of SoCs (System on Chip) dedicated to this task. In this work, several DCT approximations were evaluated in the context of image compression. The Lengwehasatit-Ortega approximation, which showed the best performance, along with the exact DCT, were implemented in VHDL and synthesized in FPGA. It was observed that, following the same design philosophy, the approximation consumed significantly fewer hardware resources, as expected.

Keywords: DCT; DCT Aproximation;VHDL; FPGA; Digital Signal Processing; Image Compression.

SUMÁRIO

1 INTRODUÇÃO

A transmissão e o processamento de imagens, tanto estáticas quanto em vídeo, estão entre as principais tarefas executadas por sistemas computacionais de uso geral nos dias atuais. Para tratar o grande volume de dados que as imagens contêm, diversas formas de codificação são empregadas e estudadas, buscando atingir altas taxas de compressão com a menor perda de qualidade possível.

A compressão, neste contexto, pode ser entendida como o uso de técnicas que utilizam redundância ou codificações para representar uma determinada informação com uma quantidade menor de dados que em sua forma não comprimida. Podemos dividir a compressão em dois tipos: com e sem perdas. Na compressão sem perdas, a natureza da informação é considerada para utilizar códigos mais eficientes, como o código de Huffman (Lathi, 2010), e eliminar informações redundantes. Já na compressão com perdas, são eliminadas informações menos relevantes, o que causa a perda da reversibilidade do processo, resultando em uma diminuição da qualidade da imagem quando descomprimida, em relação à original (Taubman, 2002).

Geralmente, os programas do tipo codificadores-decodificadores (codecs) mais eficientes utilizam uma combinação de técnicas de compressão com e sem perdas. Entre os codecs que utilizam ambas as técnicas, podemos citar o JPEG (*Joint Photographic Experts Group*). No contexto da compressão com perdas, uma das ferramentas mais utilizadas é a Transformada Discreta do Cosseno (DCT, do inglês *Discrete Cosine Transform*), que é aplicada a sinais de tempo discreto ou amostrados. A DCT é uma ferramenta matemática que concentra a maior parte da energia de um sinal nas regiões de baixa frequência, tornando-a muito útil em aplicações de compressão de arquivos.

Apesar de sua eficiência em aplicações de compressão, a DCT apresenta como ponto negativo sua complexidade matemática. Devido a isso, foram propostas na literatura diversas aproximações que buscam reduzir sua complexidade computacional, como a de Lengwehasatit e Ortega (2004).

Atualmente, a alta demanda por processamento de imagens tem levado a soluções que utilizam processadores digitais de sinais dedicados a imagens ou aceleram o processamento de imagens por meio de ASICs (*Application Specific Integrated Circuits*) ou componentes de SoC (*System on Chip*). Esses componentes processam, diretamente em hardware, matrizes ou outros cálculos necessários para o tratamento desse tipo de dado. Nesse contexto, torna-se interessante avaliar implementações em *hardware* para a DCT que reduzam o custo computacional e permitam

um menor custo de área e tempo de processamento. Este trabalho avalia a DCT exata e aproximações propostas na literatura, com o objetivo de apresentar matematicamente essas técnicas, projetar e verificar suas implementações seguindo o fluxo de *design* ASIC, e analisar a quantidade de blocos lógicos usados e o desempenho alcançado.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Realizar a implementação em *hardware* e comparar a eficiência da DCT e as aproximações como a de Lengwehasatit-Ortega em termos de consumo de blocos lógicos, como registradores e multiplicadores, por meio de simulações e análises detalhadas.

1.1.2 Objetivos específicos

- Avaliar e selecionar uma aproximação da DCT para ser implementada em *Hardware*.
- Implementar em VHDL a Transformada Discreta do Cosseno (DCT) 8x8 exata.
- Implementar em VHDL a aproximação selecionada da DCT 8x8.
- Realizar a síntese e a implementação dos circuitos em um ambiente de desenvolvimento para FPGA (*Field Programmable Gate Array*).

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados o conceito da Transformada Discreta do Cosseno e suas aplicações em processamento digital de sinais, aproximações para a Transformada Discreta do Cosseno encontradas na literatura, assim como uma introdução a compressão de imagens e *design* de sistemas digitais em *Field Programmable Gate Array* (FPGA).

2.1 TRANSFORMADA DISCRETA DO COSSENO

A Transformada Discreta do Cosseno (DCT) é uma operação matemática em tempo discreto que transforma uma função ou sinal do domínio do tempo ou espacial para o domínio da frequência. Similar à conhecida Transformada Discreta de Fourier, a DCT segue a mesma forma geral (Oppenheim, 2013):

$$X(k)=\sum_{n=0}^{N-1} x[n]\varphi_k^*[n]\,, \tag{1}$$

$$x(n)=\frac{1}{N}\sum_{n=0}^{N-1} X[k]\varphi_k^*[n]\,, \tag{2}$$

em que $\varphi_k^*[n]$ são as sequências de base, sendo essas sequências ortogonais entre si, $x[n]$ é o sinal no domínio do tempo discreto *n, N* o total de amostras e $X[k]$ o sinal transformado. Na Transformada Discreta de Fourier essas sequências de base são formadas por exponenciais complexas periódicas do tipo:

$$e^{j2\pi kn/N}\quad . \tag{3}$$

Devido a isso os sinais *X(k)* são geralmente complexos mesmo que no domínio do tempo *x(n)* seja puramente real. Entre as transformadas que apresentam sequências de base reais que também seja ortogonais entre si temos a Transformada Discreta do Cosseno, que como dito anteriormente segue a mesma forma da Transformada Discreta de Fourier mas que tem sequências de base cossenoidais.

Existem algumas formas de definir a DCT, sendo a mais comum e nosso objeto de estudo a chamada DCT-2, que para fins de conveniência de notação chamaremos apenas de DCT. A DCT-2 direta e inversa podem ser definidas da seguinte forma (Oppenheim, 2013):

$$X[k]=2\sum_{n=0}^{N-1} x[n]\cos\left(\frac{\pi k(2n+1)}{2N}\right)\,,\ 0\le k\le N-1\quad , \tag{4}$$

$$x[n]=\frac{1}{N}\sum_{K=0}^{N-1}\beta[k]X[k]\cos\left(\frac{\pi k(2n+1)}{2N}\right),\ 0\leq n\leq N-1\quad , \tag{5}$$

$$\beta[k]=\begin{cases}\frac{1}{2}\ \text{para}\ k=0\\ 1\ \text{para}\ 1\leq k\leq N-1\end{cases}\quad , \tag{6}$$

em que, a equação 4 é a transformada direta e a equação 5 é a transformada inversa que necessita da função de ponderação *β[k]*.

A DCT é amplamente empregada no processamento digital de imagens e em diversas aplicações de compressão de dados devido à sua capacidade de compactação de energia (Oppenheim, 2013). Essa propriedade significa que, ao aplicar a DCT a um sinal discreto, a maior parte da energia do sinal é concentrada nos coeficientes de baixa frequência. Como resultado, os coeficientes de alta frequência, que geralmente contêm menos energia, podem ser descartados com mínima perda na qualidade geral do sinal. Essa característica torna a DCT particularmente eficaz na redução da quantidade de dados necessários para representar uma imagem, permitindo compressão eficiente sem comprometer significativamente a fidelidade visual.

2.2 COMPRESSÃO DE IMAGENS

Compressão de imagens é a aplicação de técnicas de processamento que tem como objetivo a redução do custo de armazenamento. Essa compressão pode ser dada de diversas maneiras, sendo basicamente técnicas que levam a compressão com ou sem perdas.

A compressão sem perdas preserva a totalidade da informação original, aproveitando-se de propriedades estatísticas e redundâncias dos dados. Entre as técnicas mais conhecidas, destaca-se a codificação de entropia do código de Huffman (Lathi, 2010). Esse algoritmo atribui representações de tamanhos diferentes a cada símbolo, com base em sua probabilidade de ocorrência. Em geral, quanto maior a frequência de um símbolo, menor será o tamanho em bits de sua codificação.

Outra técnica amplamente utilizada é a codificação *Run-Length Encoding* (RLE) (Taubman, 2002), que agrupa símbolos consecutivos repetidos em um único símbolo acompanhado por um número que representa a quantidade de repetições. Por exemplo, a sequência de dados "hhhhkkkkkkbbb" pode ser representada como "4h7k3b" usando RLE.

A compressão com perdas é empregada em aplicações onde uma pequena degradação na qualidade é aceitável em troca de uma significativa redução no custo de armazenamento ou na taxa

de bits necessária para transmissão ou armazenamento. Uma técnica comum de compressão com perdas é a subamostragem de croma, que se baseia na menor sensibilidade da visão humana às variações de cor em comparação com as variações de brilho. Na subamostragem de croma, utilizada em sistemas de imagem e vídeo digitais no espaço de cores YCbCr, a luminância (Y) é codificada com mais precisão (maior número de bits) do que as componentes de crominância azul (Cb) e vermelha (Cr), permitindo uma compressão perceptualmente eficiente.

A técnica mais utilizada para compressão com perdas é a codificação por transformação, frequentemente utilizando a DCT. Como discutido anteriormente, a DCT possui a propriedade de compactação de energia, concentrando a maior parte da energia do sinal nos coeficientes de baixa frequência. Essa propriedade, por si só, não causa perdas, mas permite a aplicação de uma etapa de quantização, onde os coeficientes de alta frequência são substituídos por zeros. É essa etapa de quantização que introduz perdas de informação, permitindo, entretanto, uma compressão significativa dos dados.

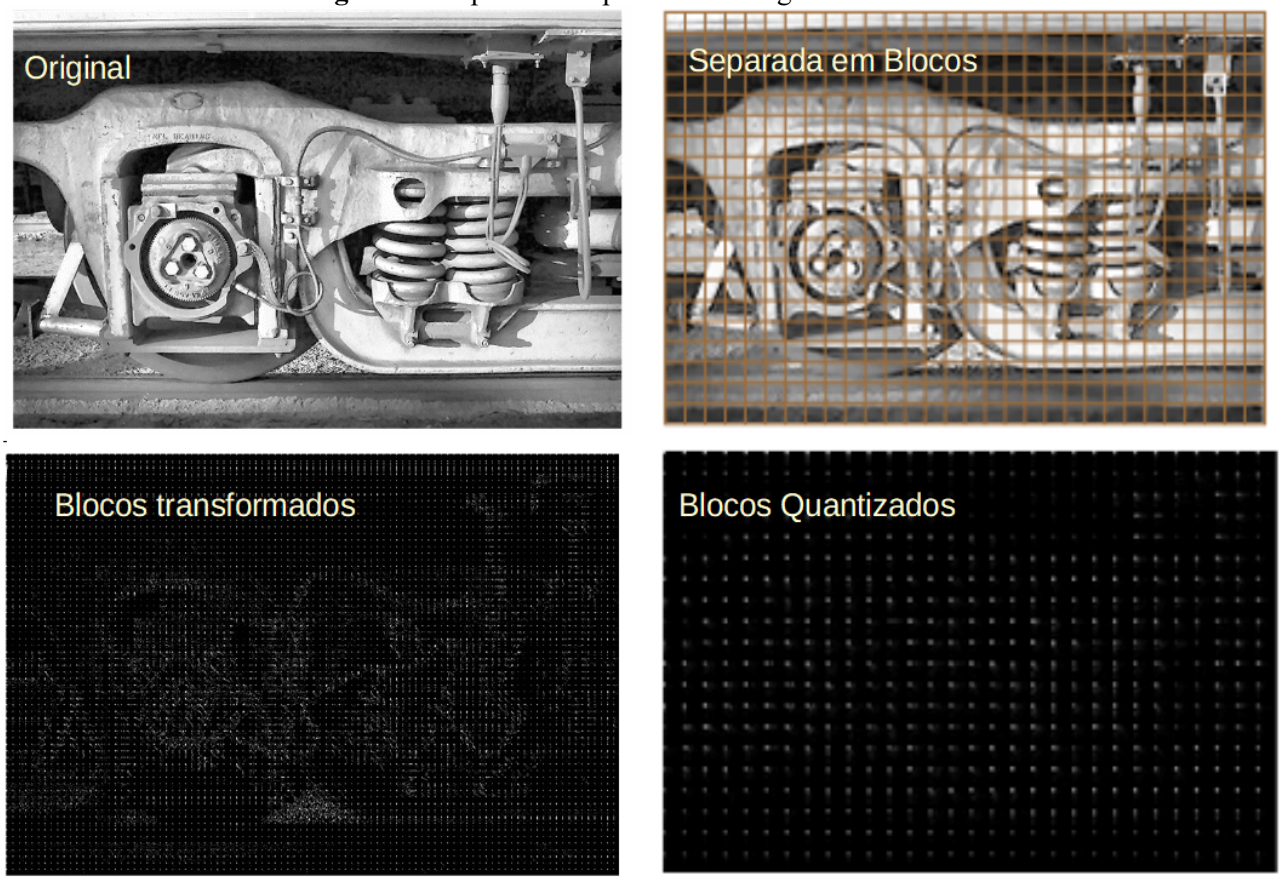
Em geral, os métodos mais utilizados aplicam técnicas de compressão sem perdas em conjunto com técnicas com perdas, como no caso dos padrões JPEG (*Joint Photographic Experts Group*) e MPEG (*Moving Picture Experts Group*).

Uma codificação no padrão JPEG segue uma sequência de passos, começando com a conversão do espaço de cores de RGB (*Red, Green, Blue*) para YCbCr e aplicando uma subamostragem de croma. Em seguida, os pixels de cada um dos três canais são divididos em blocos de tamanho 8x8. Esses blocos são transformados de forma independente para o domínio da frequência através da multiplicação pela matriz de transformação de tamanho 8x8 da DCT.

Os blocos 8x8 já transformados no domínio da frequência são então submetidos à etapa de quantização, onde os coeficientes de alta frequência são zerados ou têm seus valores drasticamente reduzidos. Como resultado da quantização, as matrizes apresentam uma grande quantidade de zeros, permitindo a aplicação eficaz da técnica RLE. Após a RLE, é feita a codificação por entropia utilizando o código de Huffman, resultando em um arquivo codificado no padrão JPEG.

Na figura 1 podem ser vistos os passos da separação em blocos de uma imagem, a DCT aplicada aos blocos e finalmente a imagem com os blocos quantizados assim como ocorre na codificação pelo padrão JPEG.

Figura 1: Etapas da compressão de imagens com JPEG.



Fonte: Adaptada de: https://whydomath.org/node/wavlets/basicjpg.html

Neste trabalho, será empregada uma aplicação simplificada da DCT em compressão de imagens, com o objetivo de avaliar apenas a etapa de cálculo da DCT em função das aproximações. O algoritmo simplificado foi aplicado na compressão de imagens em preto e branco, ou seja, que possuem somente um canal. Dentro do algoritmo, a imagem é separada em blocos 8x8, nos quais a DCT é aplicada, multiplicando-os por uma matriz 8x8 de transformação que apresenta a seguinte forma no caso da DCT exata:

$$C_N = \begin{bmatrix} 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0.3536 \\ 0{,}4904 & 0{,}4157 & 0{,}2778 & 0{,}0975 & -0{,}0975 & -0{,}2778 & -0{,}4157 & -0{,}4904 \\ 0{,}4619 & 0{,}1913 & -0{,}1913 & -0{,}4619 & -0{,}4619 & -0{,}1913 & 0{,}1913 & 0{,}4619 \\ 0{,}4157 & -0{,}0975 & -0{,}4904 & -0{,}2778 & 0{,}2778 & 0{,}4904 & 0{,}0975 & -0{,}4157 \\ 0{,}3536 & -0{,}3536 & -0{,}3536 & 0{,}3536 & 0{,}3536 & -0{,}3536 & -0{,}3536 & 0{,}3536 \\ 0{,}2778 & -0{,}4904 & 0{,}0975 & 0{,}4157 & -0{,}4157 & -0{,}0975 & 0{,}4904 & -0{,}2778 \\ 0{,}1913 & -0{,}4619 & 0{,}4619 & -0{,}1913 & -0{,}1913 & 0{,}4619 & -0{,}4619 & 0{,}1913 \\ 0{,}0975 & -0{,}2778 & 0{,}4157 & -0{,}4904 & 0{,}4904 & -0{,}4157 & 0{,}2778 & -0{,}0975 \end{bmatrix} . \quad (7)$$

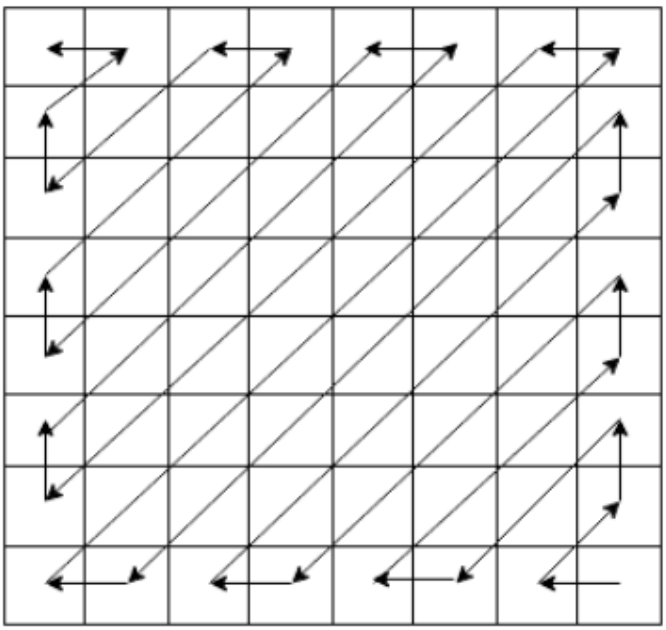
Por se tratar de um sinal bidimensional, a DCT é aplicada tanto nas linhas quanto nas colunas dos blocos da imagem. Essa operação é realizada multiplicando o produto do bloco de entrada pela DCT com a matriz de transformação transposta, de acordo com (Coutinho, 2018):

$$Y = C_N \cdot X \cdot C_N^T \quad , \tag{8}$$

em que Y é a matriz transformada, X é a matriz de entrada proveniente do bloco 8x8 e C_N a matriz de transformação da DCT como apresentada na equação 7.

Com a imagem transformada para o domínio da frequência, o algoritmo JPEG aplica uma etapa de quantização, a etapa de quantização em geral realiza uma divisão elemento a elemento entre os elementos do bloco 8x8 transformado e uma matriz de quantização pré-definida e, em seguida, arredonda os valores para números inteiros, nesse processo de arredondamento os valores pequenos são aproximados por 0. Como a DCT em si já cria uma grande quantidade de valores próximos de zero, sendo esses valores pequenos na maioria dos casos concentrados em altas frequências a versão simplificada do algoritmo de compressão zera os componentes dos blocos 8x8 transformados em sequência unidimensional ordenada como na figura 2, a quantidade de coeficientes zerados foi a variável usada no eixo das abscissas para gerar os gráficos das métricas como os das figuras 6 e 7.

Figura 2: Sentido em que os coeficientes são zerados nos blocos 8x8 durante a quantização no algoritmo.



Fonte: Adaptada de: https://www.baeldung.com/cs/jpeg-compression

Após a quantização, o algoritmo aplica a DCT inversa nas linhas e colunas de forma análoga à transformação direta, obtendo na saída a imagem comprimida com perdas, que varia conforme a

quantidade de coeficientes zerados durante a quantização. O algoritmo de compressão simplificado desenvolvido no *software* Matlab pode ser visto na íntegra no apêndice A.

Um ponto importante a ser notado é que a matriz da DCT possui em seus elementos números fracionários, que, implicam basicamente em divisões. Essas operações são realizadas diversas vezes em aplicações de compressão, quanto maior o tamanho da imagem maior a quantidade de blocos 8x8 a serem transformados e, consequentemente mais operações matemáticas de divisão são realizadas. Em termos de consumo de recursos de *hardware*, operações de divisão são geralmente mais custosas do que simples adições ou multiplicações triviais, uma vez que nem todo *hardware* tem circuito dedicado para divisões.

2.3 APROXIMAÇÕES DA DCT

Para resolver o problema do consumo de recursos de *hardware* no cálculo da DCT, diversas aproximações foram propostas na literatura. Estas aproximações utilizam mais multiplicações e operações de deslocamento, minimizando o número de operações com números fracionários. Um aspecto interessante dessas aproximações é a utilização de coeficientes aproximados por potências de 2 (negativas ou positivas), o que permite substituir operações de multiplicação/divisão por simples deslocamentos (à esquerda para multiplicar e à direita para dividir). Além disso, aproximar os coeficientes por 1, -1 ou 0 também é útil, pois representam multiplicações triviais, facilitando o cálculo. Esse enfoque permite reduzir significativamente a complexidade computacional e o uso de recursos, como unidades de divisão e multiplicadores dedicados, que são mais custosos em termos de *hardware*. Essas aproximações tornam a implementação da DCT mais eficiente e viável para aplicações em sistemas embarcados e dispositivos com recursos limitados.

Em 2001 uma aproximação baseada na aplicação da operação da função *signum* na DCT exata foi proposta por (Haweel, 2001), a chamada *Signed Discrete Cosine Transform (SDCT)* pode ser definida na forma de matriz de transformação 8x8 como a matriz C_N das equações 7 e 8 calculada a partir do produto de duas matrizes sendo elas:

$$T_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}, \tag{9}$$

$$S_8 = \begin{bmatrix} \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} \end{bmatrix}, \tag{10}$$

Outra aproximação com a proposta de usar técnicas para reduzir a complexidade da DCT foi apresentada por Lengwehasatit e Ortega (2004), chamaremos essa aproximação de LODCT e podemos definir sua matriz de transformação C_N como o produto das matrizes T_8 e S_8:

$$T_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & -1 & -1 & -1 \\ 1 & 1/2 & -1/2 & -1 & -1 & -1/2 & 1/2 & 1 \\ 1 & 0 & -1 & -1 & 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\ 1/2 & -1 & 1 & -1/2 & -1/2 & 1 & -1 & 1/2 \\ 0 & -1 & 1 & -1 & 1 & -1 & 1 & 0 \end{bmatrix}, \tag{11}$$

$$S_8 = \begin{bmatrix} \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{5}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{6}} \end{bmatrix}, \quad (12)$$

Outras aproximações encontradas na literatura são a série de aproximações que chamaremos de BAS 2008 (Bouguezel; Ahmad; Swamy, 2008), BAS 2009 (Bouguezel; Ahmad; Swamy, 2009) e BAS 2013 (Bouguezel; Ahmad; Swamy; SWAMY, 2013). A BAS 2013, em particular, é uma DCT binária que corresponde a *Walsh-Hadamard transform* (WHT) (Bouguezel; Ahmad; Swamy, 2013). De maneira análoga a definição das matrizes de transformação 8x8 da SDCT e da LODCT podemos definir a BAS 2008 8x8 como o produto das matrizes (13) e (14), respectivamente:

$$T_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 \\ 1 & 1/2 & -1/2 & -1 & -1 & -1/2 & 1/2 & 1 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1/2 & -1 & 1 & -1/2 & -1/2 & 1 & -1 & 1/2 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad (13)$$

$$S_8 = \begin{bmatrix} \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{5}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}, \quad (14)$$

A matriz de transformação 8x8 da aproximação BAS 2009 é obtida pelo produto das seguintes matrizes T_8 e S_8, (15) e (16), respectivamente:

$$T_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad (15)$$

$$S_8 = \begin{bmatrix} \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}, \tag{16}$$

Finalmente, a matriz de transformação 8x8 da aproximação BAS 2013 é calculada a partir do produto de (17) e (18):

$$T_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}, \tag{17}$$

$$S_8 = \begin{matrix} \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{8}} \end{matrix}, \quad (18)$$

Podemos notar das equações (9), (11), (13), (15) e (17) que as matrizes T_8 são compostas por elementos unitários, divisões por 2 ou zeros, que representam baixa complexidade computacional, estando os números fracionários contidos apenas nas matrizes diagonais S_8. Essa configuração garante que as aproximações sejam muito menos custosas computacionalmente do que a DCT enquanto se aproximam de suas propriedades matemáticas.

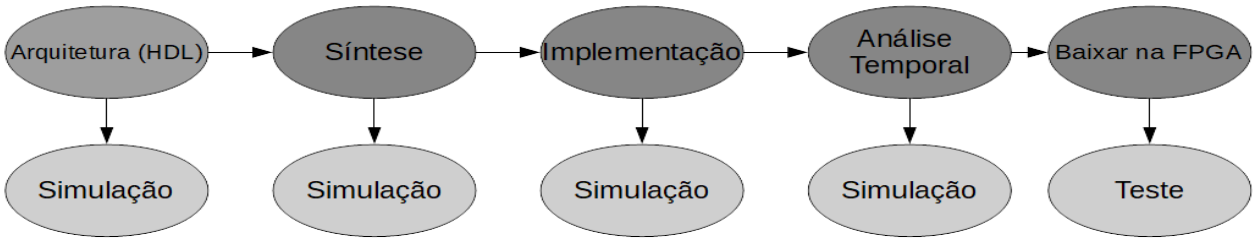
2.4 IMPLEMENTAÇÃO EM *HARDWARE*

A implementação em *hardware*, como a realizada em um ASIC, é desenvolvida com o objetivo de atender a um propósito específico. No entanto, ao contrário dos ASICs, a lógica de uma FPGA é desenvolvida pelo engenheiro de projeto e pode ser modificada a qualquer momento, inclusive com o produto já em campo. Isso confere uma flexibilidade significativa às FPGAs, tornando-as ideais para prototipagem e aplicações que possam necessitar de atualizações ou modificações após a implantação inicial.

A grande vantagem de utilizar FPGAs para projetos de menor escala ou testes reside no fato de que, embora apresentem um custo unitário maior comparado aos ASICs, as FPGAs oferecem um fluxo de *design* significativamente mais curto. Como pode ser visto na figura 3, o fluxo de *design* em FPGAs envolve uma quantidade muito menor de etapas, permitindo que o design seja revisado e reconfigurado rapidamente, resultando em um tempo total de engenharia reduzido. Em contraste, o fluxo de projeto para ASICs requer uma quantidade muito maior de etapas (Rao, 2023) como pode

ser visto na figura 4, incluindo a implementação física, o que acarreta altos custos de fabricação e um tempo de desenvolvimento substancialmente mais longo.

Figura 3: Fluxo de Design FPGA.

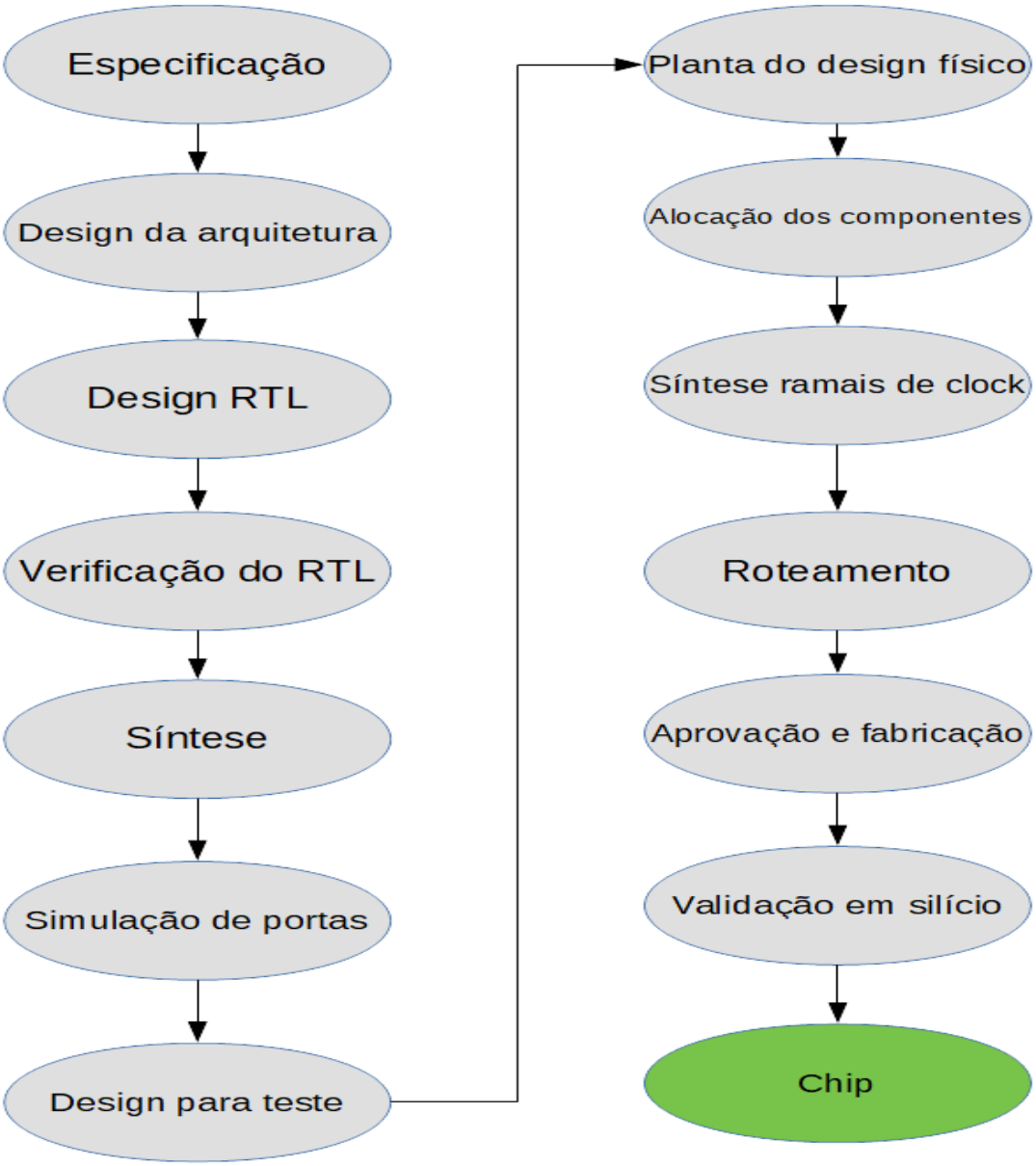


Fonte: O autor, 2024.

FPGAs pertencem à categoria de dispositivos de lógica programável e são compostas internamente por matrizes de portas lógicas, organizadas em módulos de lógica adaptativa (ALMs). Além dessas portas lógicas, as FPGAs incluem blocos dedicados, como processadores de sinal digital (DSPs) e memórias de acesso aleatório (RAM). Esses componentes, associado a capacidade de configurar o roteamento entre eles, permitem ao projetista desenvolver a lógica desejada de maneira eficiente e personalizada (Intel, 2024).

A flexibilidade e a capacidade de reconfiguração das FPGAs fazem delas uma escolha ideal para a implementação e comparação de diferentes algoritmos de processamento de sinais, como a Transformada Discreta do Cosseno (DCT) exata e suas aproximações. As FPGAs permitem uma avaliação detalhada e prática do custo de lógica digital associado a cada abordagem, possibilitando ajustes e otimizações em tempo real, sem a necessidade de custos adicionais de fabricação que seriam inevitáveis com ASICs.

Figura 4: Fluxo de Design ASIC.



Fonte: O autor, 2024

3 METODOLOGIA

Nesse trabalho, realizou-se uma avaliação da DCT e algumas de suas aproximações no contexto de compressão de imagens em preto e branco, onde a aproximação que melhor desempenhou foi implementada em hardware, assim como, a DCT com o objetivo de verificar a quantidade de recursos de *hardware* utilizados.

Inicialmente, foi realizada uma busca na literatura de algumas das aproximações para a DCT disponíveis, além disso, foi realizado um estudo sobre compressão de imagens e como a DCT é aplicada nesse contexto e quais métricas poderiam ser usadas para avaliar a perda da qualidade em imagens comprimidas.

A DCT e as aproximações foram comparadas quanto a qualidade da imagem comprimida em termos das métricas SSIM e PSNR aplicando um algoritmo simplificado de compressão com o objetivo de selecionar a aproximação que teve um desempenho mais próximo da DCT exata. Para tal foi usado um banco de imagens clássicas por motivos de simplicidade.

Por fim a DCT exata e a aproximação que melhor desempenhou foram implementadas em FPGA utilizando a linguagem VHDL para verificar de forma quantitativa em termos de *hardware* a economia de usar uma aproximação ao invés da DCT exata.

4 RESULTADOS

4.1 AVALIAÇÃO DAS APROXIMAÇÕES

Para selecionar a melhor aproximação dentre as diversas existentes na literatura, um algoritmo de compressão simplificado foi desenvolvido para realizar a compressão com perdas em imagens em preto e branco. A qualidade das imagens comprimidas foi medida quantitativamente utilizando os valores de SSIM (*Structural Similarity Index*) e PSNR (*Peak Signal to Noise Ratio*). Essas métricas foram calculadas para as compressões realizadas com a DCT exata, SDCT, LODCT, BAS 2008, BAS 2009, BAS 2013 e WHT.

PSNR é uma métrica amplamente utilizada na literatura para avaliar a qualidade de imagens comprimidas. Ela calcula a relação sinal-ruído com base no erro quadrático médio (EQM), fornecendo uma medida quantitativa da diferença entre a imagem original e a imagem comprimida, onde quanto maior seu valor maior a qualidade da imagem. A fórmula para calcular a PSNR é apresentada a seguir (C-Y. Chen; C-H. Chen; C-Ho. Chen; K-P. Lin, 2016):

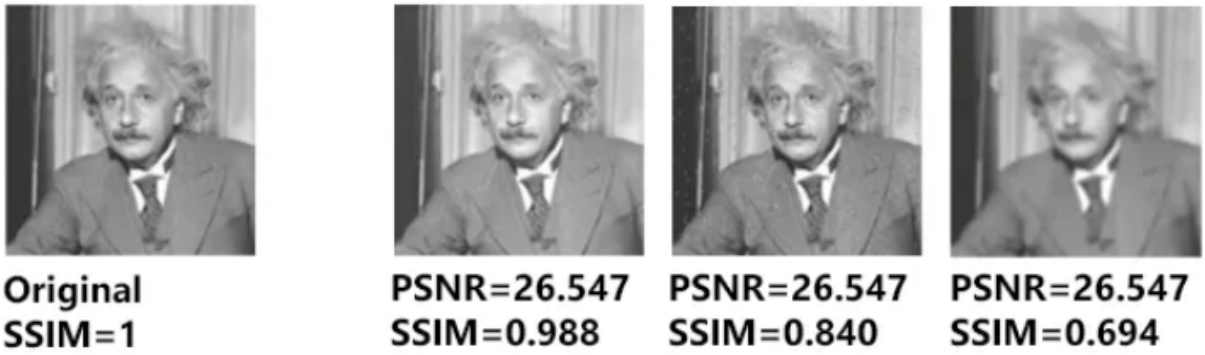
$$EQM = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} [\, p(i,j) - pt(i,j)^k]^2 \quad , \tag{19}$$

$$PSNR = 10 \log_{10} \left(\frac{255^2}{EQM} \right) \quad , \tag{20}$$

em que $m \times n$ é a dimensão da imagem, $p(i,j)$ é o pixel original e $pt(i,j)^k$ é o k-ésimo pixel transformado.

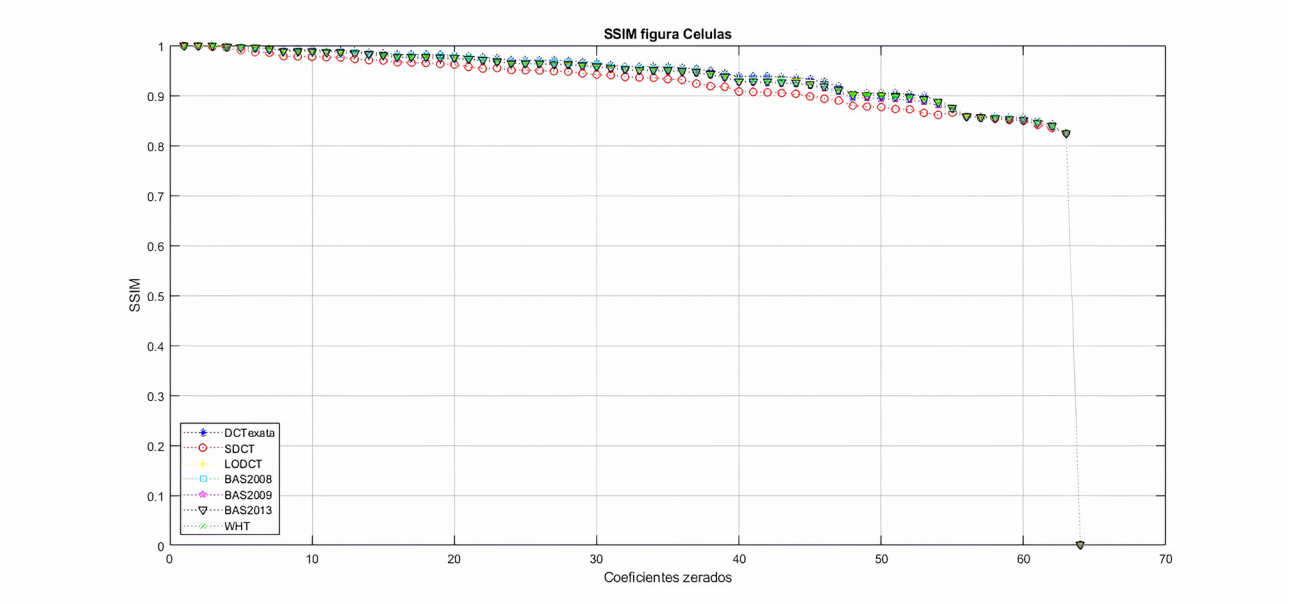
A SSIM é introduzida para contornar uma limitação da PSNR, que é a insensibilidade à percepção humana de qualidade. Enquanto a PSNR mede a relação sinal-ruído considerando o erro quadrático médio, a SSIM avalia a qualidade da imagem levando em conta a luminância, contraste e estrutura, aspectos mais alinhados com a percepção visual humana. Como ilustrado na Figura 5, o ruído pode se concentrar em regiões da imagem que não interferem significativamente na qualidade percebida. Considerando o Sinal X como o sinal original, a SSIM mede a qualidade do Sinal Y, refletindo a similaridade entre as imagens com base em luminância, contraste e estrutura (Wang, 2004), onde quanto maior seu valor numérico maior a similaridade entre a imagem original e a comprimida.

Figura 5: Impacto da posição do ruído na PSNR e SSIM.



Fonte: https://medium.com/@datamonsters/a-quick-overview-of-methods-to-measure-the-similarity-between-images-f907166694ee

Figura 6: SSIM em função da quantidade de coeficientes zerados por bloco para a figura células presente no canto superior esquerdo da figura 8.

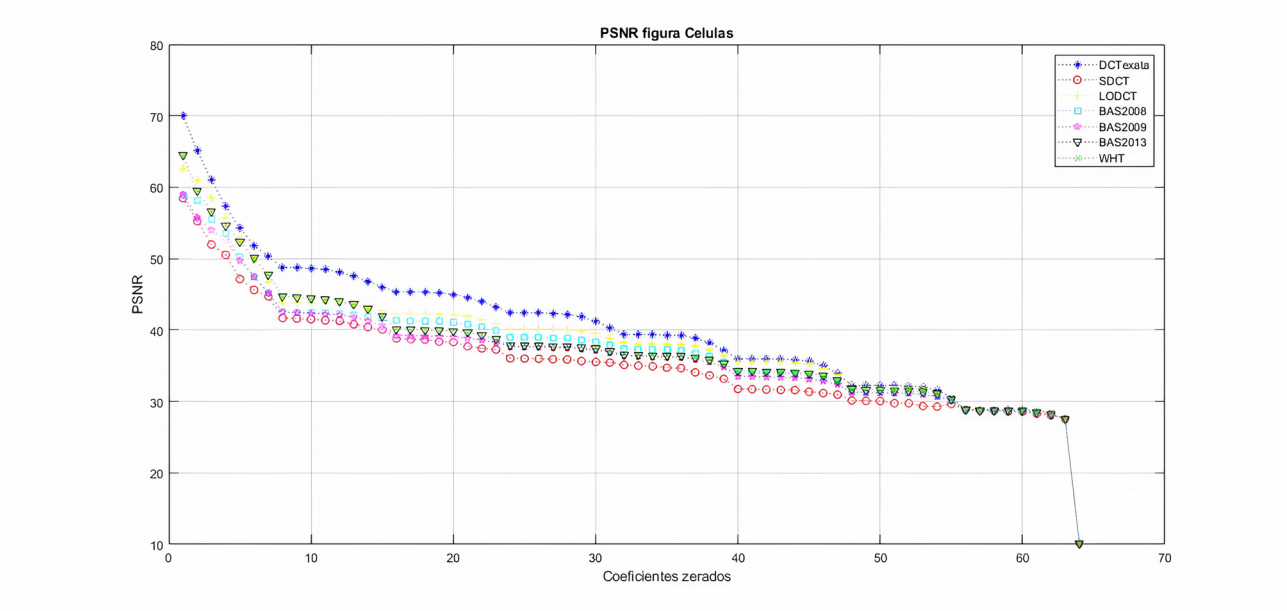


Fonte: O autor, 2023.

De forma a facilitar a visualização e a análise dos resultados foram plotados gráficos como os das figuras 6 e 7, onde o valor da SSIM e PSNR aparecem em função da quantidade de elementos que foram zerados na matriz transformada dentro do algoritmo. Esses gráficos foram gerados para as 5 imagens que podem ser vistas na figura 8, e por fim, a área abaixo de cada uma das curvas geradas para as DCTs da SSIM e PSNR foram calculadas e compiladas nas tabelas 1 e 2 respectivamente, onde para cada uma das imagens o melhor resultado está em negrito e o segundo

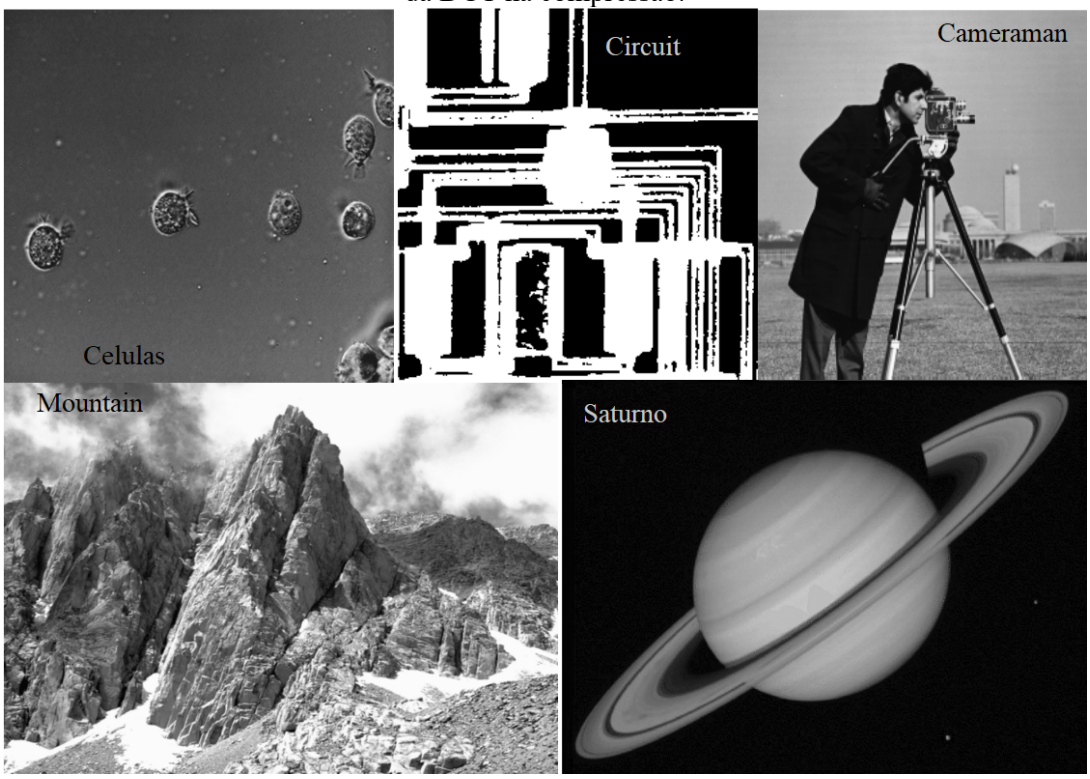
melhor em itálico. Da tabela 1 podemos observar que a LODCT teve o melhor resultado para 3 das 5 imagens e o segundo melhor para a imagem *circuit*, e apesar de não ter o melhor desempenho na imagem saturno ainda apresentou resultado muito satisfatório com valor de SSIM na casa de 57 assim como a própria DCT exata. Nos valores de PSNR da tabela 2 a LODCT obteve o melhor resultado em 4 das 5 imagens e o segundo melhor na imagem de saturno. A partir desses resultados concluímos então que no contexto de compressão em que foram aplicadas na maioria dos casos a LODCT foi a aproximação que melhor desempenhou e, por isso, foi a escolhida para ser implementada em *hardware*, assim como a DCT exata.

Figura 7: PSNR em função da quantidade de coeficientes zerados por bloco para a figura células presente no canto superior esquerdo da figura 8.



Fonte: O autor, 2023.

Figura 8: Imagens em preto e branco usadas para avaliar o desempenho das aproximações da DCT na compressão.



Fonte: Adaptado de: https://people.math.sc.edu/Burkardt/data/tif/tif.html

Tabela 1: Valores da integral abaixo da curva da SSIM para cada uma das figuras e das DCTs.

DCT	celulas	circuit	cameraman	mountain	saturno
Exata	59,0625	54,0132	56,1615	55,3197	57,2841
SDCT	57,8893	51,0219	54,206	52,4635	55,5743
LODCT	**58,9172**	*53,3098*	**55,7912**	**54,7459**	57,0224
BAS 2008	58,7383	52,855	*55,559*	54,0056	**57,0722**
BAS 2009	58,6274	52,8006	55,4163	53,6512	*57,029*
BAS 2013	*58,7438*	**53,3778**	55,5174	*54,0958*	56,8233
WHT	58,7438	53,3778	55,5174	54,0958	56,8233

Fonte: O autor, 2024.

Tabela 2: Valores da integral abaixo da curva da PSNR para cada uma das figuras e das DCTs.

DCT	células	circuit	cameraman	mountain	saturno
Exata	2,54E+03	1,28E+03	1,98E+03	2,01E+03	2,47E+03
SDCT	2,24E+03	1,11E+03	1,76E+03	1,74E+03	2,27E+03
LODCT	**2,43E+03**	**1,25E+03**	**1,91E+03**	**1,90E+03**	*2,42E+03*
BAS 2008	2,36E+03	1,20E+03	1,87E+03	1,82E+03	**2,43E+03**
BAS 2009	2,31E+03	1,19E+03	1,84E+03	1,79E+03	2,41E+03
BAS 2013	*2,37E+03*	*1,23E+03*	*1,87E+03*	*1,86E+03*	2,38E+03
WHT	2,37E+03	1,23E+03	1,87E+03	1,86E+03	2,38E+03

Fonte: O autor, 2024.

4.2 IMPLEMENTAÇÃO EM *HARDWARE*

O *design* proposto no trabalho contempla apenas a implementação propriamente dita das matrizes de transformação 8x8, não foram levados em consideração os circuitos acessórios como unidades de controle e memória, pois o foco se deu em analisar a economia de recursos de *hardware* obtida pela utilização da aproximação.

4.2.1 Proposta de *Design* para a DCT exata

Para a criação do *design* da lógica da DCT exata partiu-se da análise da multiplicação de matrizes entre um vetor coluna de entrada e a matriz de transformação como segue:

$$Y = T_8 \times X \quad , \tag{21}$$

$$Y = \begin{bmatrix} 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 \\ 0{,}4904 & 0{,}4157 & 0{,}2778 & 0{,}0975 & -0{,}0975 & -0{,}2778 & -0{,}4157 & -0{,}4904 \\ 0{,}4619 & 0{,}1913 & -0{,}1913 & -0{,}4619 & -0{,}4619 & -0{,}1913 & 0{,}1913 & 0{,}4619 \\ 0{,}4157 & -0{,}0975 & -0{,}4904 & -0{,}2778 & 0{,}2778 & 0{,}4904 & 0{,}0975 & -0{,}4157 \\ 0{,}3536 & -0{,}3536 & -0{,}3536 & 0{,}3536 & 0{,}3536 & -0{,}3536 & -0{,}3536 & 0{,}3536 \\ 0{,}2778 & -0{,}4904 & 0{,}0975 & 0{,}4157 & -0{,}4157 & -0{,}0975 & 0{,}4904 & -0{,}2778 \\ 0{,}1913 & -0{,}4619 & 0{,}4619 & -0{,}1913 & -0{,}1913 & 0{,}4619 & -0{,}4619 & 0{,}1913 \\ 0{,}0975 & -0{,}2778 & 0{,}4157 & -0{,}4904 & 0{,}4904 & -0{,}4157 & 0{,}2778 & -0{,}0975 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \quad , \tag{22}$$

onde T_8 é a matriz de transformação da DCT, X o vetor entrada e Y o vetor de saída. Observa-se que a matriz de transformação da DCT exata é composta por elementos fracionários como pode ser visto na equação 22, o que implica a necessidade de implementar circuitos divisores em *hardware* para realizar essa operação. No entanto, foi proposta uma abordagem alternativa de aproximações de divisões baseadas em multiplicações e deslocamentos, que serão discutidas em detalhe mais adiante.

O *design* foi desenvolvido e testado no ambiente de simulação ModelSim 10.5b da Mentor Graphics, utilizando a linguagem de descrição de *hardware* VHDL. Conforme observado na equação 22, além de números fracionários, temos números com sinal envolvidos na operação. Para lidar com isso, utilizamos a biblioteca *ieee.numeric_std*. Esta biblioteca permite a representação dos elementos da DCT como vetores do tipo *signed* (com sinal), além de permitir operações aritméticas envolvendo esses vetores.

Para representar os números fracionários, foram usados vetores do tipo *signed* de 16 bits, onde os 8 bits menos significativos representam a parte fracionária e os 8 bits mais significativos a parte inteira. Dentro do algoritmo em VHDL, a vírgula foi omitida, restando apenas a operação com números inteiros. Finalmente, esse número inteiro é multiplicado pela entrada, também do tipo

signed de 16 bits, e o resultado é deslocado 8 bits à direita, gerando assim uma aproximação para a divisão composta por operações mais simples de multiplicação e deslocamento. Os números inteiros usados para representar os elementos da matriz de transformação da DCT foram obtidos multiplicando seus valores decimais por 2^8. Tomando como exemplo o primeiro elemento da matriz 0,3536 conforme a equação a seguir:

$$0{,}3536 \times 2^8 = 90{,}5216 \quad , \tag{23}$$

em seguida, o resultado, que é um número na base decimal, foi convertido para a base binária aproximando a parte fracionária em 8 bits:

$$90{,}5216_d = 01011010{,}10000101_b \quad , \tag{24}$$

por fim, omitimos a vírgula do resultado e convertemos novamente da base binária para decimal:

$$0101101010000101_b = 23173_d \quad , \tag{25}$$

esse método foi repetido para todos os elementos T_{ij} da matriz de transformação e os resultados foram compilados na tabela 3. Os valores originais podem ser obtidos fazendo o caminho contrário a partir dos valores finais, pois trata-se de um método reversível.

Tabela 3: Conversão dos coeficientes da DCT para a representação de números fracionários adotada.

Valor original (T_{ij})	**$T_{ij} . 2^8$**	**Binário com vírgula**	**Valor final**
0,3536	90,5216	01011010,10000101	23173
0,4904	125,5424	01111101,10001010	32138
0,4157	106,4192	01101010,01101011	27243
0,2778	71,1168	01000111,00011101	18205
0,0975	24,96	00011000,11110101	6389
0,1913	48,9728	00110000,11111001	12537
0,4619	118,2464	01110110,00111111	30271

Fonte: O autor. 2024.

A entidade em VHDL foi desenvolvida contando com 8 entradas do tipo *signed* de 16 bits, e uma entrada para um sinal de *clock*, como os elementos da matriz de saída são resultados de multiplicações envolvendo vetores de 16 bits, as 8 saídas da entidade da DCT foram declaradas como vetores tipo *signed* de 32 bits. Com a entidade do *design* definida, a arquitetura foi desenvolvida a partir da multiplicação de matrizes onde cada elemento da matriz de saída é o resultado da soma das multiplicações da linha correspondente pelo vetor coluna de entrada, usando os valores obtidos na tabela 3 e os deslocamentos de 8 bits a direita discutidos anteriormente. O código completo da implementação pode ser visto no apêndice B.

O *design* desenvolvido foi submetido a um *testbench*, que consiste em aplicar uma série de sinais de entrada ao circuito e observar se a saída reflete o comportamento esperado. Para validar a lógica implementada, foi realizada a multiplicação da matriz de transformação da DCT por vetores coluna arbitrários via Matlab. Esses mesmos valores dos vetores coluna foram então aplicados como estímulos na entrada do *design* dentro do ModelSim.

Entre os valores testados, destacamos o caso em que todos os valores do vetor de entrada são 255 (8 bits). O comportamento esperado para este cenário é apresentado a seguir, onde a saída do circuito deve corresponder aos resultados teóricos calculados:

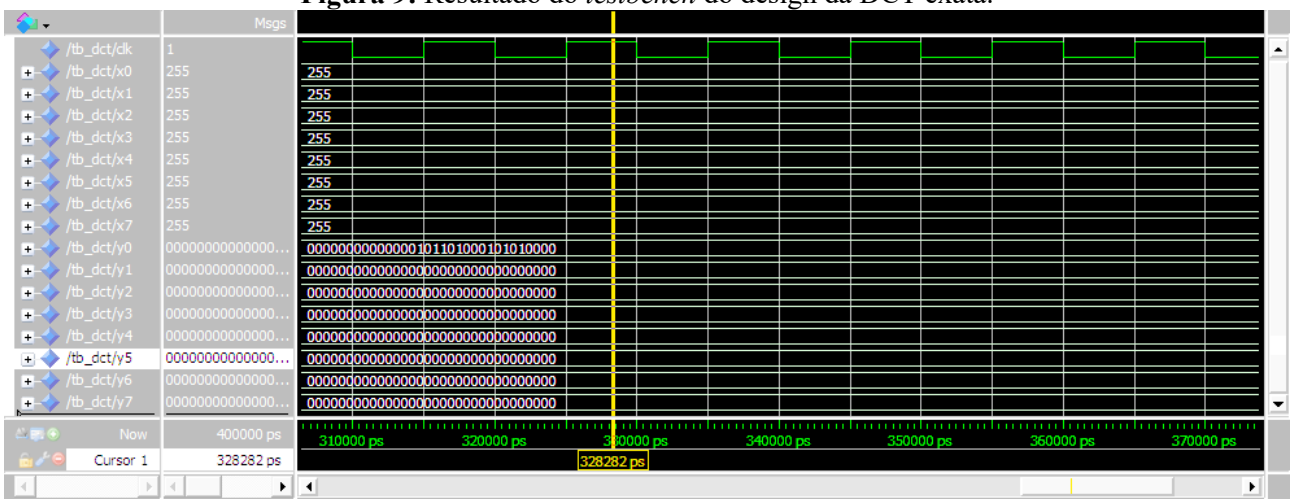
$$\begin{bmatrix} 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 & 0{,}3536 \\ 0{,}4904 & 0{,}4157 & 0{,}2778 & 0{,}0975 & -0{,}0975 & -0{,}2778 & -0{,}4157 & -0{,}4904 \\ 0{,}4619 & 0{,}1913 & -0{,}1913 & -0{,}4619 & -0{,}4619 & -0{,}1913 & 0{,}1913 & 0{,}4619 \\ 0{,}4157 & -0{,}0975 & -0{,}4904 & -0{,}2778 & 0{,}2778 & 0{,}4904 & 0{,}0975 & -0{,}4157 \\ 0{,}3536 & -0{,}3536 & -0{,}3536 & 0{,}3536 & 0{,}3536 & -0{,}3536 & -0{,}3536 & 0{,}3536 \\ 0{,}2778 & -0{,}4904 & 0{,}0975 & 0{,}4157 & -0{,}4157 & -0{,}0975 & 0{,}4904 & -0{,}2778 \\ 0{,}1913 & -0{,}4619 & 0{,}4619 & -0{,}1913 & -0{,}1913 & 0{,}4619 & -0{,}4619 & 0{,}1913 \\ 0{,}0975 & -0{,}2778 & 0{,}4157 & -0{,}4904 & 0{,}4904 & -0{,}4157 & 0{,}2778 & -0{,}0975 \end{bmatrix} \times \begin{bmatrix} 255 \\ 255 \\ 255 \\ 255 \\ 255 \\ 255 \\ 255 \\ 255 \end{bmatrix} = \begin{bmatrix} 721{,}2489 \\ 0 \\ -0 \\ 0 \\ 0 \\ -0 \\ -0 \\ 0 \end{bmatrix} . \quad (26)$$

Essa entrada foi aplicada como estímulo no *testbench* do *design* no ModelSim e o resultado pode ser visto na figura 9, onde o valor 1011010000101010000 aparece como saída em y_0. Para analisar se esse valor está coerente aplicamos a mesma lógica utilizada para representar números fracionários, com os 8 bits menos significativos representando os decimais e, em seguida, convertendo da base binária para decimal tanto a parte inteira quanto a parte fracionária obtendo:

$$1011010001{,}01010000_b = 721{,}3125_d \quad . \quad (17)$$

Podemos notar que o resultado obtido no *testbench* indica que o design desenvolvido é, de fato, uma implementação simplificada em *hardware* da DCT exata.

Figura 9: Resultado do *testbench* do design da DCT exata.



Fonte: O autor, 2024.

Com o *design* devidamente testado e aprovado, a próxima etapa da implementação consistiu em submeter o código VHDL para a análise e síntese em FPGA, realizada no Quartus Prime Lite. Como o *design* proposto consistiu apenas na matriz de transformação propriamente dita, o programa atribuiu as entradas e saídas da entidade diretamente a pinos de propósito geral disponível na FPGA. Portanto, foi necessário escolher uma placa que possuísse um grande número de pinos desse tipo disponíveis para o usuário. Optamos pela FPGA *Cyclone V* 5CGXFC9E6F35C7.

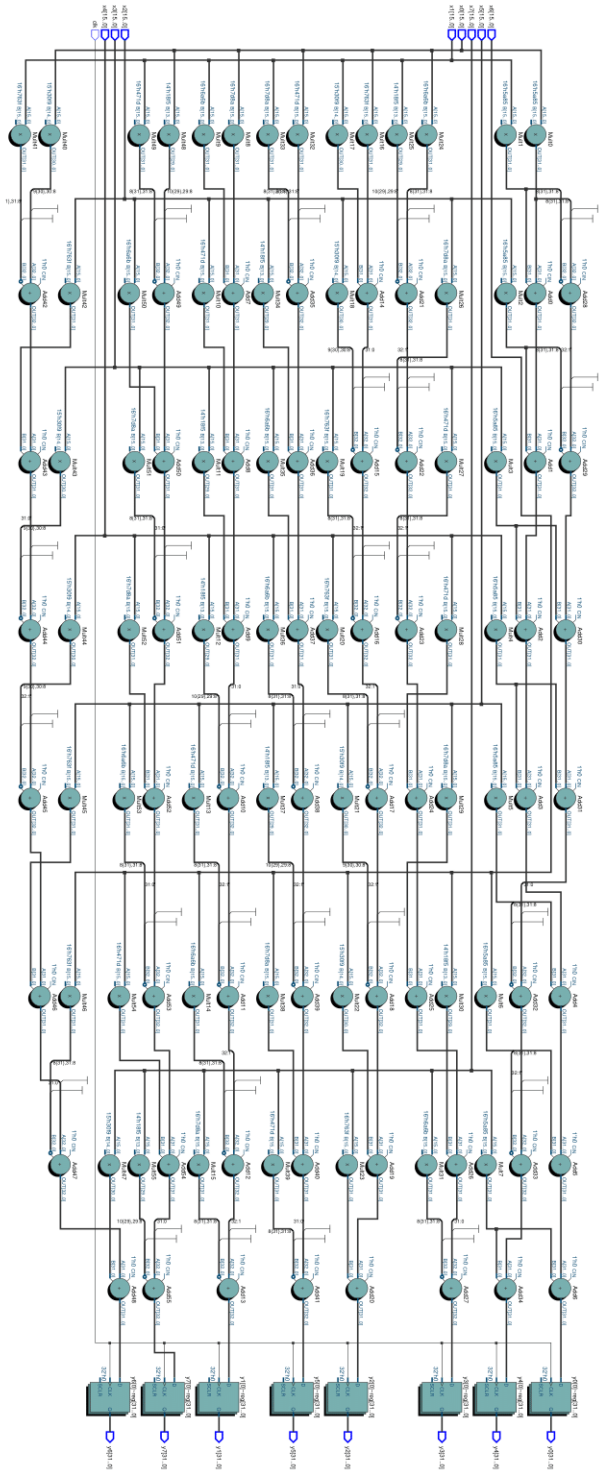
Após a análise e síntese, o Quartus Prime Lite permite a visualização do circuito em nível de lógica de registradores (*RTL Viewer*), conforme mostrado na Figura 10. Além disso, o Quartus Prime Lite gera relatórios detalhados que informam as quantidades de recursos de *hardware* da FPGA utilizados para implementar a lógica. As informações mais pertinentes para nossa discussão foram compiladas na Tabela 4.

Tabela 4: Recursos de *hardware* da FPGA utilizados para implementar o design da DCT exata.

Recursos	**Uso**
Registradores Lógicos Dedicados	256
Uso de ALUT Combinacional para lógica	904
Pinos de E/S	385 / 616
Total de blocos de DSP	56 / 342
Utilização de lógica (ALMs necessários / total de ALMs no dispositivo)	574 / 113,560

Fonte: O autor, 2024.

Figura 10: Visão RTL do design desenvolvido para a DCT exata.



Fonte: O autor, 2024

Da Tabela 4, podemos observar que foram utilizados muitos pinos de entrada/saída, totalizando 385. Além disso, foram empregados 574 ALMs (*Adaptive Logic Modules*), que são os componentes básicos da família *Cyclone V*. Cada ALM pode suportar até oito entradas e oito saídas. O *design* utilizou 904 ALUTs (*Adaptive Look-Up Tables*). Uma *Look-Up Table* é uma matriz de dados que mapeia valores de entrada para valores de saída, aproximando assim uma função matemática. Uma ALUT é uma construção lógica que representa o que pode ser implementado pelo *hardware* lógico combinacional de um ALM. Também foram utilizados 256 registradores dedicados e 56 DSPs (*Digital Signal Processors*). Os blocos DSP contêm registradores de deslocamento de entrada para implementar aplicações de filtros digitais, incluindo filtros *Finite Impulse Response* (FIR) e *Infinite Impulse Response* (IIR), e multiplicadores. Os multiplicadores no bloco DSP podem opcionalmente alimentar um somador/subtrator ou acumulador dentro do bloco. No contexto do nosso *design*, os DSPs representam a quantidade de multiplicações realizadas na transformada, sendo muito importantes para determinar a eficiência da aplicação. A partir do RTL da figura 10 foram contabilizados 56 multiplicadores e 56 somadores utilizados no *design*.

4.2.2 Proposta de Design para a LODCT

A implementação em *hardware* da LODCT seguiu a mesma lógica da DCT exata, com a diferença de que a aproximação tem sua matriz de transformação dividida em duas partes, conforme demonstrado nas equações 11 e 12. Dessa forma, o produto da matriz T_8 da equação 11 pelo vetor de entrada foi implementado e posteriormente chamado como componente no *design* final, que multiplicou esse produto pela matriz diagonal S_8 (equação 12) da LODCT. O código completo da implementação pode ser visto no apêndice C.

De forma semelhante ao design anterior, o circuito foi submetido a um *testbench*. Desta vez, destacamos a entrada arbitrária [40,40,10,10,10,40,40,40], com o resultado esperado de [81,3173, −12,2474, 33,5410, 12,2474, −10,6066, 0, 0, −12,2474]. Os sinais de saída do circuito podem ser vistos na Figura 11, e os valores da saída foram convertidos no mesmo padrão de representação de números fracionários proposto, com os 8 bits menos significativos representando a parte decimal e o restante como a parte inteira. As conversões da saída foram compiladas na Tabela 5.

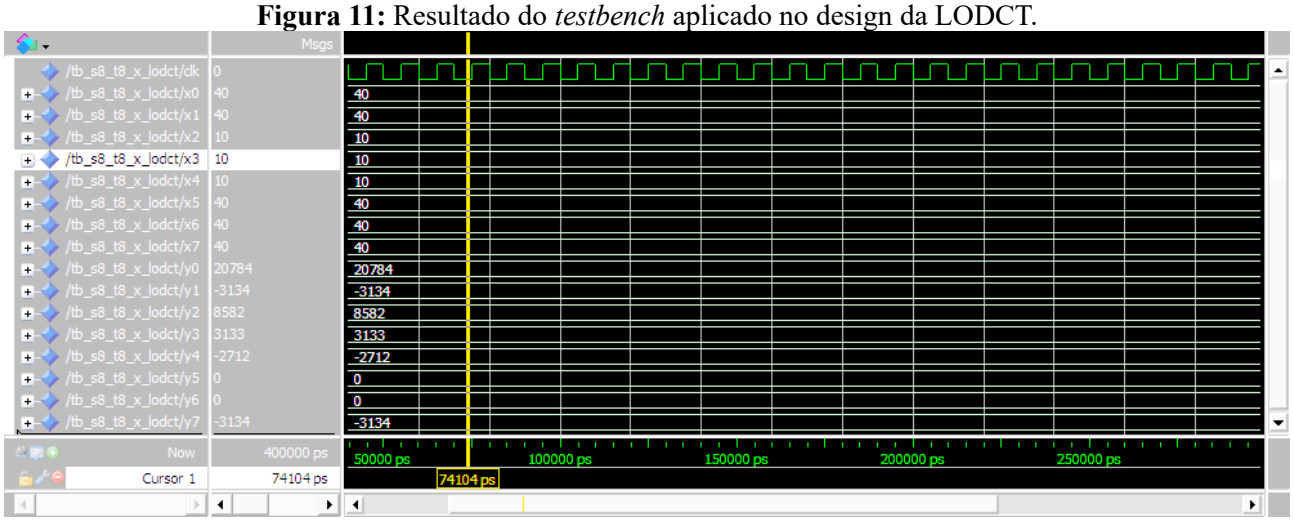
A partir do resultado do *testbench*, podemos concluir que o código em VHDL desenvolvido implementa, de fato, a LODCT.

Tabela 5: Resultados do *testbench* convertidos para forma decimal convencional seguindo padrão de representação de números fracionários proposto.

Saída	Saida em binário	Saída convertida	Saída esperada
20784	101001,00110000	81,1875	81,3173
-3134	1100,00111110	-12,2421875	-12,2474
8582	100001,10000110	33,5234375	33,541
3133	1100,00111101	12,23828125	12,2474
-2712	1010,10011000	-10,59375	-10,6066
0	0	0	0
0	0	0	0
-3134	1100,00111110	-12,2421875	-12,2474

Fonte: o autor, 2024.

Figura 11: Resultado do *testbench* aplicado no design da LODCT.



Fonte: O autor, 2024.

Com o código testado e aprovado o *design* foi submetido a análise e síntese no Quartus Prime Lite, utilizando a mesma FPGA da DCT exata. A visualização do circuito em RTL pode ser vista na figura 12 e os recursos utilizado para implementar a LODCT foram compilados na tabela 6.

Podemos notar na Tabela 6 que a LODCT utilizou o mesmo número de registradores dedicados e pinos de entrada/saída que a DCT exata, uma vez que a entidade foi escrita de forma semelhante, com mudanças apenas na arquitetura. Foram usadas 453 ALUTs contra 904 da DCT exata, além de apenas 268 ALMs na LODCT em contraste com os 574 da DCT. O maior ganho, no entanto, foi na redução do número de DSPs, de 56 para 8, além de uma redução de 56 para 47 somadores como pode ser visto no RTL da figura 12. Isso se deve ao fato de a aproximação utilizar apenas 8 multiplicações provenientes da matriz diagonal, enquanto o restante das operações se resume a somas, subtrações e deslocamentos. Esse resultado revela o caráter da aproximação de reduzir significativamente a quantidade de recursos de *hardware* necessários para implementá-la,

traduzindo-se em ganho de desempenho e redução da área de silício necessária para sua confecção em circuitos integrados.

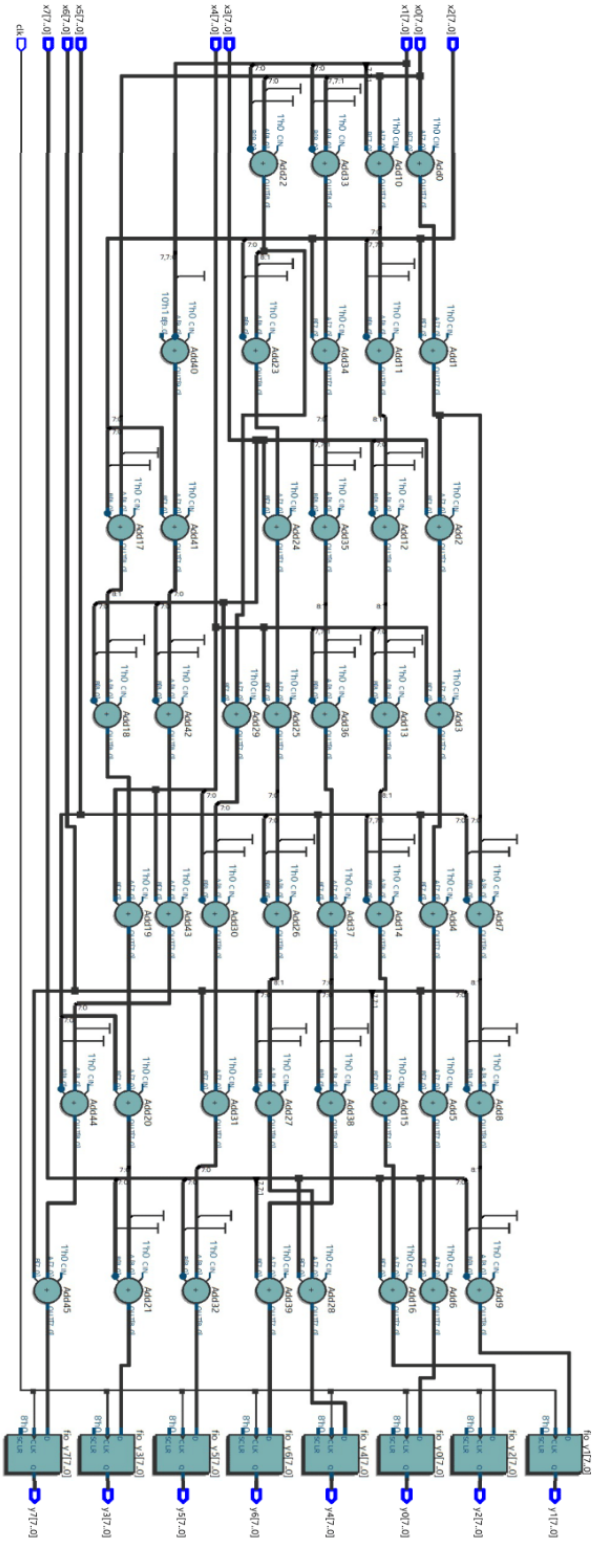
Um detalhe que vale a pena ser comentado é que os *designs* propostos não consideraram questões temporais, como tempo de propagação dos sinais, tempo de *setup* e tempo de *hold*. Esses aspectos devem ser analisados minuciosamente em trabalhos futuros, caso se deseje levar o design à implementação prática propriamente dita.

Tabela 6: Recursos da FPGA utilizados para implementar a lógica da LODCT.

Recursos	**Uso**
Registradores Lógicos Dedicados	256
Uso de ALUT Combinacional para lógica	453
Pinos de E/S	385 / 616
Total de blocos de DSP	8 / 342
Utilização de lógica (ALMs necessários / total de ALMs no dispositivo)	268 / 113,560

Fonte: O autor, 2024.

Figura 12: Visão em RTL do design para a LODCT desenvolvido.



Fonte: O autor, 2024

5 CONCLUSÃO

Neste projeto foram avaliadas a DCT e algumas aproximações que se propõem a consumir menos recursos de *hardware* que a versão exata. Para verificar isso, as aproximações foram comparadas em aplicações de compressão de imagem, e a aproximação de Lengwehasatit-Ortega (LODCT) que mais se aproximou da DCT exata em termo de qualidade foi a escolhida para ser implementada em *hardware.*

Um *design* em VHDL foi desenvolvido para sintetizar a lógica da matriz de transformação 8x8 da DCT e da aproximação LODCT. Para verificar a lógica, foram realizadas verificações por meio de *testbenchs,* que atestaram que o design implementado correspondeu ao correto funcionamento das transformadas. O código em VHDL foi sintetizado em uma FPGA modelo Cyclone V 5CGXFC9E6F35C7, utilizando o Quartus Prime 18.1. Após executar a análise e síntese, observou-se que a DCT exata consumiu 56 blocos de DSPs, 56 somadores, 256 registradores dedicados e 904 ALUTs, enquanto a LODCT utilizou apenas 8 DSPs, 47 somadores, também 256 registradores dedicados e 453 ALUTs.

Como esperado a aproximação consumiu significativamente menos recursos da FPGA, principalmente no que se refere aos DSPs que representam os multiplicadores dedicados utilizados. Finalmente, o trabalho demonstrou de forma quantitativa que as aproximações são excelentes candidatas a serem utilizadas em aplicações que necessitem aplicar a DCT, mas que disponham de limitações de hardware e busquem alto desempenho e baixo consumo de energia, como nos sistemas embarcados e portáteis atuais.

REFERÊNCIAS

BOUGUEZEL, S.; AHMAD, M. O.; SWAMY, M. N. S. Low-complexity 8×8 transform for image compression. **Electronics Letters**, Argélia, v. 44, n. 21, p. 1249 – 1250, 2008. Disponível em: https://www.researchgate.net/publication/224338452_Low-complexity_88_transform_for_image_compression. Acesso em: 11 jun. 2024.

BOUGUEZEL, S.; AHMAD, M. O.; SWAMY, M. N. S. A fast 8×8 transform for image compression. In: International Conference on Microelectronics, 2009, Marrocos. **Anais** [...] Marrocos: Institute of Electrical and Eletronics Engineers, 2009, p. 74-77, v. 1, n. 1. Disponível em: https://digital-library.theiet.org/content/journals/10.1049/el_20082239. Acesso em: 11 jun. 2024.

BOUGUEZEL, S.; AHMAD, M. O.; SWAMY, M. N. S. Binary discrete cosine and Hartley transforms. **IEEE Transactions on Circuits and Systems I Regular Papers**, Nova Iorque, v. 60, n. 4, p. 989-1002, 2013. Disponível em: https://ieeexplore.ieee.org/document/6375773/. Acesso em: 11 jun. 2024.

CHEN, C.-Y., CHEN, C.-H., CHEN, C.-H., & LIN, K.-P. An automatic filtering convergence method for iterative impulse noise filters based on PSNR checking and filtered pixels detection. **Expert Systems With Applications**, Jordânia, v. 63, n. 1, p. 198-207, 2016. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0957417416303487?via%3Dihub. Acesso em: 12 jun. 2024.

COUTINHO, V. A. **Aproximações de baixa complexidade para transformadas discretas multidimensionais**. Tese (Doutorado em Engenharia Elétrica) - Universidade Federal de Pernambuco, Recife, 2018. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/34022/1/TESE%20V%C3%ADtor%20de%20Andrade%20Coutinho.pdf. Acesso em: 12 jun. 2024.

HAWEEL, T. I. A new square wave transform based on the DCT. **Signal Processing**, Holanda, v. 81, n. 1, p. 2309-2319, 2001. Disponível em: https://www.researchgate.net/profile/Tarek-Haweel/publication/223850489_A_new_square_wave_transform_based_on_the_DCT/links/5a966452aca27214056961e2/A-new-square-wave-transform-based-on-the-DCT.pdf. Acesso em: 12 jun. 2024.

INTEL. **FPGA basics and getting started**. Santa Clara, Califórnia, EUA: Intel. Online, Disponível em: https://www.intel.com/content/www/us/en/support/programmable/supportresources/fpga-training/getting-started.html. Acesso em: 11/06/2024.

LATHI. B. P. **Sistemas de comunicações analógicos e digitais modernos**. 4. ed. Rio de Janeiro: LTC, 2012. Disponível em: https://www.academia.edu/27338039/Sistemas_de_Comunica%C3%A7%C3%B5es_Anal%C3%B3gicos_e_Digitais_Modernos_Lathi_4a_Ed_pdf. Acesso em: 13 jun. 2024.

LENGWEHASATIT, K.; ORTEGA, A. Scalable variable complexity approximate forward DCT. **IEEE Transactions on Circuits and Systems for Video Technology**, California, v. 14, n. 11, p. 1236-1248, 2004. Disponível em: https://www.researchgate.net/profile/Antonio-Ortega-2/publication/3308702_Scalable_Variable_Complexity_Approximate_Forward_DCT/links/54d595cb0cf246475807dafd/Scalable-Variable-Complexity-Approximate-Forward-DCT.pdf. Acesso em: 13 jun. 2024.

OPPENHEIM, A. V.; SCHAFER, R. W. **Processamento em tempo discreto de sinais**. 3. ed. São Paulo: Pearson Education do Brasil, 2012.

RAO, Ravi. **ASIC vs FPGA**: a comprehensive comparison. [*S. l.*]: Wevolver, 09 ago. 2023. Artigo. Disponível em: https://www.wevolver.com/article/asic-vs-fpga. Acesso em: 05 abr. 2024.

TAUBMAN, David S.; MARCELLIN, Michael W. **JPEG2000**: image compression fundamentals, standards and practice. Nova York: Springer, 2002. Disponível em: https://www.hlevkin.com/hlevkin/04imageprocDoc/David%20S.%20Taubman,%20%20Michael%20W.%20Marcellin%20%20(auth.)%20JPEG2000%20Image%20Compression%20Fundamentals,%20Standards%20and%20Practice%20%202002.pdf. Acesso em: 13 jun. 2024.

APÊNDICE A – ALGORITMO DE COMPRESSÃO SIMPLIFICADO DESENVOLVIDO NO MATLAB

```
%Implementação da DCT(Discrete Cosine Transform) em processamento de blocos
%8x8 de imagem preto e branco simples para compressão com perdas utilizando
%máscara matricial simples para zerar componentes de alta frequência
%Autor: Leonardo Nogueira Lindolfo da Silva

I = imread('C:\Users\leoli\Downloads\mountain.tif');
I = im2double(I);

T = dctmtx(8);
dct = @(block_struct) T*block_struct.data*T';%Definição da função dct que é
definida como o produto matricial
%entre a matriz 8x8 da DCT e o bloco da imagem e o resultado desse produto
%é novamente multiplicado pela matriz 8x8 da DCT a fim de obter a DCT aplicada
%nas linhas e colunas do bloco da imagem
B = blockproc(I,[8 8],dct);%Imagem reconstruída utilizando os blocos 8x8 no
domínio
%transformado
mask = [1   1   1   1   1   1   1   1
        1   1   1   1   1   1   1   1
        1   1   1   1   1   1   1   1
        1   1   1   1   1   1   1   1
        1   1   1   1   1   1   1   1
        1   1   1   1   1   1   1   1
        1   1   1   1   1   1   1   1
        1   1   1   1   1   1   1   1];%Mascara para definir os componentes de
frequência zerados para a
    %compressão

indice_resultado = 1;

for i=8:-1:1
    for j = 8:-1:1

        mask(i,j)=0;

        compress = @(block_struct) mask.*block_struct.data;%Definição da função
de compressão que aplica a máscara
        %em cada bloco 8x8 da imagem no dominio transformado
        B2 = blockproc(B,[8 8], compress);%construção da imagem comprimida no
domínio transformado
        invdct = @(block_struct) T'*block_struct.data*T;%função inversa
aplicando propriedade de que a matriz inversa
        %é a transversa
        I2 = blockproc(B2,[8 8],invdct);%construção da imagem no domínio
original comprimida

        resultadossim(indice_resultado) = ssim(I2,I);
        resultadopsnr(indice_resultado) = psnr(I2,I);
        indice_resultado = indice_resultado + 1;
    end
end
integralssimDCT = trapz(x,resultadossim);
integralpsnrDCT = trapz(x,resultadopsnr);
```

APÊNDICE B – *DESIGN* DA DCT EXATA EM VHDL

```
--DCT exata
--Autor: Leonardo Nogueira Lindolfo da Silva

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DCT is
      port(clk: in std_logic;
      x0,x1,x2,x3,x4,x5,x6,x7: in signed(15 downto 0);
      y0,y1,y2,y3,y4,y5,y6,y7: out signed(31 downto 0));
end DCT;

architecture behav_DCT of DCT is

begin
process(clk)
begin
 if rising_edge(clk) then
y0 <=
shift_right(x0*23173,8)+shift_right(x1*23173,8)+shift_right(x2*23173,8)+shift_ri
ght(x3*23173,8)+
shift_right(x4*23173,8)+shift_right(x5*23173,8)+shift_right(x6*23173,8)+shift_ri
ght(x7*23173,8);

y1 <=
shift_right(x0*32138,8)+shift_right(x1*27243,8)+shift_right(x2*18205,8)+shift_ri
ght(x3*6389,8)-
shift_right(x4*6389,8)-shift_right(x5*18205,8)-shift_right(x6*27243,8)-
shift_right(x7*32138,8);

y2 <= shift_right(x0*30271,8)+shift_right(x1*12537,8)-shift_right(x2*12537,8)-
shift_right(x3*30271,8)-
shift_right(x4*30271,8)-
shift_right(x5*12537,8)+shift_right(x6*12537,8)+shift_right(x7*30271,8);
```

```
y3 <= shift_right(x0*27243,8)-shift_right(x1*6389,8)-shift_right(x2*32138,8)-
shift_right(x3*18205,8)+
shift_right(x4*18205,8)+shift_right(x5*32138,8)+shift_right(x6*6389,8)-
shift_right(x7*27243,8);

y4 <= shift_right(x0*23173,8)-shift_right(x1*23173,8)-
shift_right(x2*23173,8)+shift_right(x3*23173,8)+
shift_right(x4*23173,8)-shift_right(x5*23173,8)-
shift_right(x6*23173,8)+shift_right(x7*23173,8);

y5 <= shift_right(x0*18205,8)-
shift_right(x1*32138,8)+shift_right(x2*6389,8)+shift_right(x3*27243,8)-
shift_right(x4*27243,8)-shift_right(x5*6389,8)+shift_right(x6*32138,8)-
shift_right(x7*18205,8);

y6 <= shift_right(x0*12537,8)-shift_right(x1*30271,8)+shift_right(x2*30271,8)-
shift_right(x3*12537,8)-
shift_right(x4*12537,8)+shift_right(x5*30271,8)-
shift_right(x6*30271,8)+shift_right(x7*12537,8);

y7<= shift_right(x0*6389,8)-shift_right(x1*18205,8)+shift_right(x2*27243,8)-
shift_right(x3*32138,8)+
shift_right(x4*32138,8)-shift_right(x5*27243,8)+shift_right(x6*18205,8)-
shift_right(x7*6389,8);
    end if;

end process;
end;
```

APÊNDICE C – *DESIGN* DA LODCT EM VHDL

```
--Matriz T8 da LODCT
--Autor: Leonardo Nogueira Lindolfo da Silva

library ieee;
use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

entity LODCT is
      port(clk: in std_logic;
      x0,x1,x2,x3,x4,x5,x6,x7: in signed(15 downto 0);
      y0,y1,y2,y3,y4,y5,y6,y7: out signed(15 downto 0));
end LODCT;

architecture LODCT_behav of LODCT is



begin



m: process(clk)
begin
    if rising_edge(clk) then

--primeira saida
y0 <= x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7;
--segunda saida
y1 <= x0 + x1 + x2 - x5 - x6 - x7;
--terceira saida
y2 <= x0 + shift_right(x1, 1) - shift_right(x2, 1) - x3 - x4 - shift_right(x5,
1)+
shift_right(x6,1) + x7;
--quarta saida
y3 <= x0 - x2 - x3 + x4 + x5 - x7;
--quinta saida
y4 <= x0 - x1 - x2 + x3 + x4 - x5 - x6 + x7;
--sexta saida
```

```
y5 <= x0 - x1 + x3 - x4 + x6 - x7;
--setima saida
y6 <= shift_right(x0, 1) - x1 + x2 - shift_right(x3, 1) - shift_right(x4, 1) +
x5 - x6 + shift_right(x7, 1);
--oitava saida
y7 <= - x1 + x2 - x3 + x4 - x5 + x6;

    end if;
end process;



end;

--LODCT
--Essa entidade que implementara a LODCT de fato
-- Utilizando o componente LODCT que implementa a multiplicacao da entrada pela
matriz T8
-- o resultado dessa multiplicacao é multiplicada pela matriz S8 para obter o
resultado final transformado
--Autor: Leonardo Nogueira Lindolfo da SIlva

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity S8_T8_x_LODCT is
      port(clk: in std_logic;
      x0,x1,x2,x3,x4,x5,x6,x7: in signed(15 downto 0);
      y0,y1,y2,y3,y4,y5,y6,y7: out signed(31 downto 0));
end S8_T8_x_LODCT;

architecture LODCT_imp of S8_T8_x_LODCT is

component LODCT is
      port(clk: in std_logic;
      x0,x1,x2,x3,x4,x5,x6,x7: in signed(15 downto 0);
      y0,y1,y2,y3,y4,y5,y6,y7: out signed(15 downto 0));
end component LODCT;

signal fio_y0,fio_y1,fio_y2, fio_y3, fio_y4, fio_y5,fio_y6, fio_y7: signed(15
downto 0);
```

```
begin

c_T8: LODCT PORT MAP (clk,x0,x1,x2,x3,x4,x5,x6,x7,fio_y0,fio_y1,fio_y2, fio_y3,
fio_y4, fio_y5,fio_y6, fio_y7);

m: process(clk)
begin
    if rising_edge(clk) then
y0 <= shift_right(fio_y0*23134, 8); --equivale a multiplicar por 1/sqrt(8)
--A multiplicação foi feita de forma aproximada considerando  a parte inteira de
0,358*2^(16-8)
--E deslocando o resultado de 16-8 casas para a direita, para evitar usar
divisores
y1 <= shift_right(fio_y1*26738, 8); --equivale a multiplicar por 1/sqrt(6)
y2 <= shift_right(fio_y2*29294, 8); --equivale a multiplicar por 1/sqrt(5)
y3 <= shift_right(fio_y3*26738, 8); --equivale a multiplicar por 1/sqrt(6)
y4 <= shift_right(fio_y4*23134, 8); --equivale a multiplicar por 1/sqrt(8)
y5 <= shift_right(fio_y5*26738, 8); --equivale a multiplicar por 1/sqrt(6)
y6 <= shift_right(fio_y6*29294, 8); --equivale a multiplicar por 1/sqrt(5)
y7 <= shift_right(fio_y7*26738, 8); --equivale a multiplicar por 1/sqrt(6)


    end if;
end process;

end;
```