Sistema em SoC para compressão de imagem baseado em JPEG

Artur Miranda Oliveira

1 Introdução

Sistemas computacionais empregam sistemas gráficos extensivamente, ainda mais nos dias atuais com o crescente aumento de aplicações envolvendo reconhecimento de padrões em imagens. Como é sabido, informações guardadas e transmitidas por hardware devem atingir certas especificações, normalmente relacionadas a tempo de processamento e armazenamento limitado. Com imagens não é diferente, além disso, imagens tendem a ser especialmente grandes. Por exemplo, uma imagem com resolução de 512x512 onde cada pixel pode conter entre aproximadamente 16 milhões de cores (24 bits), tem cerca de 786.432 bytes. Esse valor é extremamente alto, ainda mais se as imagens forem transmitidas e processadas em tempo real por dispositivos IoT cujos ambientes, por vezes, tem velocidade de conexão à internet, capacidade de processamento e energia disponível limitada.

Felizmente, no mundo real, as imagens podem ser comprimidas eliminando-se redundância de informações ou utilizando métodos com perdas. Um exemplo de método com perda é o JPEG, que utiliza a DCT (discrete cosine transform) e, através dos coeficientes obtidos para cada cossenoide que descreve a imagem, escolhem-se aqueles que tem maior impacto na construção da imagem (apenas os maiores coeficientes são mantidos).

2 Representação de imagens em sistemas digitais

Antes de entrar nos detalhes da compressão de imagem, é importante introduzir como um sistema digital, como um computador ou um *System on Chip* (SoC), interpreta e processa uma imagem. Uma imagem nada mais é do que uma matriz de duas dimensões de pixels. O termo pixel é uma abreviação para *picture element* [Sal10]. Para muitos engenheiros, programadores e usuários pixels podem ser pensados como pequenos quadrados, e isso geralmente é verdade para dispositivos como monitores. Entretanto, para outros dispositivos, pixels podem ser circulares, retangulares, etc. Na definição, um pixel deve ser considerado como um um ponto sem dimensão, podendo ser discreto no caso dos sistemas de computação digitais ou pode ser um valor real, como no caso dos sensores, e.g., a saída analógica do sensor de óptico de uma câmera.

Além disso, uma imagem pode ser classificada em vários tipos, dependendo de suas características. Alguns deles são:

- **Imagem bi-nível (ou monocromática)**: É uma imagem onde os pixels podem ter um de dois valores, normalmente referidos como preto e branco. Cada pixel em tal imagem é representado por um bit, tornando este o tipo mais simples de imagem.
- **Imagem em tons de cinza**: Um pixel em tal imagem pode ter um dos valores de 0 até $n-1$, indicando um dos 2^n tons de cinza (ou tons de alguma outra cor). O valor de n normalmente é compatível com o tamanho de um byte; ou seja, é 4, 8, 12, 16, 24 ou algum outro múltiplo conveniente de 4 ou 8.

- **Imagem de tons contínuos**: Este tipo de imagem pode ter muitas cores semelhantes (ou tons de cinza). Quando pixels adjacentes diferem por apenas uma unidade, é difícil ou até impossível para o olho distinguir suas cores. Como resultado, uma imagem como essa pode conter áreas com cores que parecem variar continuamente conforme o olho se move pela área. Um pixel em tal imagem é representado por um número grande (no caso de muitos tons de cinza) ou três componentes (no caso de uma imagem colorida). Imagens de tons contínuos são normalmente imagens naturais e obtidas por meio de fotografia com uma câmera digital ou pela digitalização de uma fotografia ou pintura.

- **Imagem de tons discretos** : Normalmente, é uma imagem artificial. Pode ter poucas ou muitas cores, mas não possui o ruído e desfoque de uma imagem natural. Exemplos são um objeto artificial ou máquina, uma página de texto, um gráfico, uma animação ou o conteúdo de uma tela de computador. (Nem toda imagem artificial é de tom discreto. Uma imagem gerada por computador que se pretende natural é uma imagem de tons contínuos, apesar de ser gerada artificialmente.) Objetos artificiais, texto e desenhos a linha têm bordas nítidas e bem definidas, e são portanto altamente contrastantes em relação ao restante da imagem (o fundo). Pixels adjacentes em uma imagem de tom discreto frequentemente são idênticos ou variam significativamente em valor. Esse tipo de imagem não compacta bem com métodos com perdas, pois a perda de apenas alguns pixels pode tornar uma letra ilegível ou mudar um padrão familiar para um não reconhecível. Métodos de compressão para imagens de tons contínuos muitas vezes não lidam bem com bordas nítidas, então métodos especiais são necessários para a compressão eficiente dessas imagens. Note que uma imagem de tom discreto pode ser altamente redundante, já que o mesmo caractere ou padrão pode aparecer muitas vezes na imagem.

Para fins de ilustração, segue abaixo a representação de uma imagem monocromática e sua matriz de pixels na Figura 1 e uma representação de uma imagem de tom contínuo e sua matriz de pixels Figura 2, Figura 3.

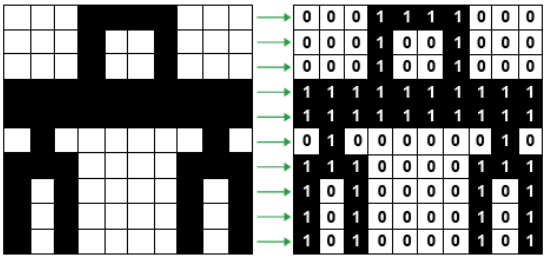

Figura 1: Imagem monocromática

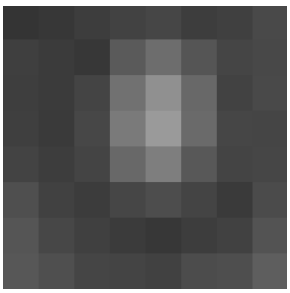

Figura 2: Imagem de tons contínuos

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

Figura 3: Matriz de pixels de imagem de tons contínuos

3 Compressão de imagem

Uma imagem, afinal, existe para que as pessoas a vejam, então, quando é compactada, é aceitável perder características da imagem às quais o olho não é sensível. Esta é uma das principais ideias por trás dos diversos métodos de compressão de imagem com perda.

De maneira geral, a informação pode ser compactada se for redundante. No entanto, com a compressão com perda, temos um novo conceito, que é a compressão removendo a irrelevância. Uma imagem pode ser compactada com perda ao remover informações irrelevantes (e.g. pequenas variações de cor que são imperceptíveis ao olho humano), mesmo que a imagem original não tenha nenhuma redundância. Neste trabalho serão considerados três métodos de compressão, que são as bases para o JPEG: *Run Length Encoding*, *Discrete Cosine Transform* (DCT) e o *Huffman Encoding*.

3.1 DCT

Transformações de imagem são projetadas para ter duas propriedades: (1) reduzir a redundância da imagem, diminuindo o tamanho da maioria dos pixels e (2) identificar as partes menos importantes da imagem, isolando as várias frequências da imagem. Intuitivamente, associamos uma frequência a uma onda. Ondas de água, ondas sonoras e ondas eletromagnéticas têm frequências, mas pixels em uma imagem também podem apresentar frequências.

Para entender esse conceito é preciso introduzir a DCT. A DCT em uma dimensão é definida por:

$$G_f = \sqrt{\frac{2}{n}} C_f \sum_{t=0}^{n-1} p_t \cos\left[\frac{(2t+1)f\pi}{2n}\right]$$

onde,

$$C_f = \begin{cases} \frac{1}{\sqrt{2}}, & f = 0, \\ 1, & f > 0, \end{cases} \quad \text{for } f = 0, 1, \ldots, n-1. \tag{1}$$

A entrada é um conjunto de n valores de dados p_t (pixels, amostras de áudio ou outros dados), e a saída é um conjunto de n coeficientes de transformação DCT (ou pesos) G_f.

Entretanto, uma maneira mais intuitiva de enxergar a DCT é interpretá-la na forma matricial. Uma transformada é definida da seguinte forma, onde uma função, vetor ou número é mapeado para outro valor, em outro domínio ou base:

$$\mathcal{T}\{f(t)\} = F(\omega)$$

de tal forma que,

$$f(t) = \mathcal{T}^{-1}\{F(\omega)\}$$

e na forma matricial,

$$\mathbf{X} = \mathbf{T}\mathbf{x}$$

onde $\mathbf{x}$ é o vetor de entrada (função ou dados originais), $\mathbf{T}$ é a matriz de transformação, e $\mathbf{X}$ é o vetor de saída (coeficientes transformados). Caso a matriz T for composta por cossenos, o vetor $\mathbf{x}$ pode ser visto como uma soma ponderada de cossenos, sendo os pesos dados por $\mathbf{X}$. Considere o exemplo onde n = 3:

$$\begin{bmatrix} G_0 \\ G_1 \\ G_2 \end{bmatrix} = \begin{bmatrix} \cos 0 & \cos 0 & \cos 0 \\ \cos \frac{\pi}{6} & \cos \frac{3\pi}{6} & \cos \frac{5\pi}{6} \\ \cos \frac{2\pi}{6} & \cos \frac{2\cdot 3\pi}{6} & \cos \frac{2\cdot 5\pi}{6} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} = \mathbf{D} \cdot \mathbf{p}. \tag{2}$$

Se os ângulos selecionados para a DCT são $\theta_j = (j + 0,5)\frac{\pi}{n}$, então os componentes de cada vetor $\mathbf{v}_k$ são $\cos\left[k\,(j+0,5)\,\frac{\pi}{n}\right]$. Cada vetor $\mathbf{v}_k$ é uma linha de $\mathbf{D}$. A equação acima pode ser escrita com as linhas da matriz $\mathbf{D}$ normalizadas por $\sqrt{3}, \sqrt{1.5}, \sqrt{1.5}$, respectivamente. Logo, $\mathbf{p}$, pode ser escrita da seguinte forma:

$$\begin{bmatrix} 0.5773 & 0.7071 & 0.4083 \\ 0.5773 & -0.0000 & -0.8165 \\ 0.5773 & -0.7071 & 0.4083 \end{bmatrix} \cdot \begin{bmatrix} G_0 \\ G_1 \\ G_2 \end{bmatrix} = \mathbf{p}$$

Ou seja, $\mathbf{p}$ é uma soma ponderada de senoides discretas de frequência 0, $\frac{\pi}{6}$, $\frac{2\pi}{6}$.

Para ilustrar mais ainda, tem-se outros três exemplos. O primeiro exemplo é $\mathbf{p} = (p, p, p)$. Os três componentes de $\mathbf{p}$ são idênticos, então eles correspondem a uma frequência zero. O produto $\mathbf{D} \cdot \mathbf{p}$ produz os coeficientes de frequência $(1.7322p, 0, 0)$, indicando ausência de altas frequências. O segundo exemplo é $\mathbf{p} = (p, 0, -p)$. Os três componentes de $\mathbf{p}$ variam lentamente de p a $-p$, então esse vetor contém uma baixa frequência. O produto $\mathbf{D} \cdot \mathbf{p}$ produz os coeficientes $(0, 1.4142p, 0)$, confirmando esse resultado. O terceiro exemplo é $\mathbf{p} = (p, -p, p)$. Os três componentes de $\mathbf{p}$ variam de p a $-p$ e depois a p, então esse vetor contém uma alta frequência. O produto $\mathbf{D} \cdot \mathbf{p}$ produz $(0, 0, 1.6329p)$, indicando novamente a frequência correta.

Para uma entrada $\mathbf{p}$ de 8 elementos a representação gráfica em pixels da DCT pode ser vista na Figura 4, onde os elementos de $\mathbf{p}$ podem ser vistos como uma soma ponderada de cada vetor $\mathbf{v}_k$, ou seja, $\mathbf{p} = \sum_i w_i \mathbf{v}_i$. A DCT, dado um vetor de entrada, retorna os pesos que cada $\mathbf{v}_k$ tem de ter para que esses vetores, quando somados ponderadamente, retornem a sequência de entrada. Os vetores $\mathbf{v}_k$ podem ser interpretados como sendo imagens base.

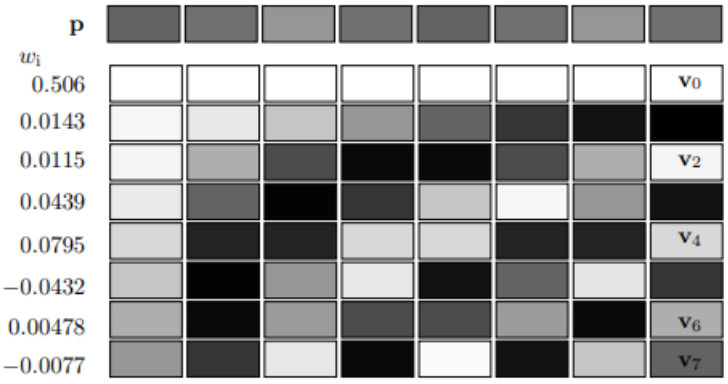

Figura 4: Representação gráfica em pixels da DCT

Para duas dimensões, as imagens bases, para uma entrada 8x8, são vistas na Figura 5.

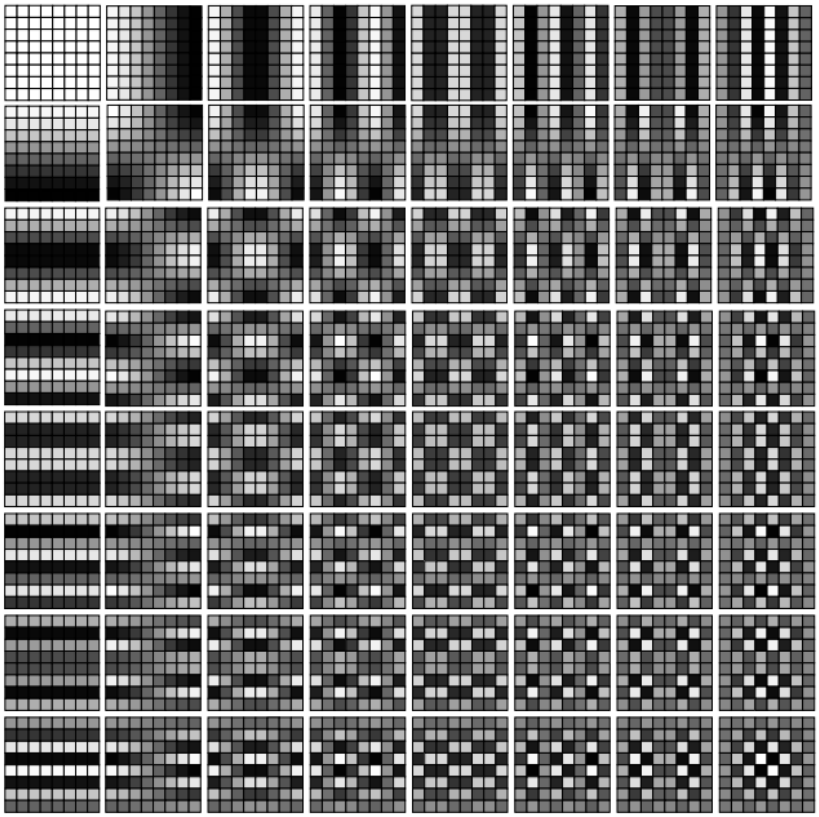

Figura 5: Imagens base para a DCT de duas dimensões

A DCT em duas dimensões para um bloco de imagem 8x8 é dada por:

$$G_{u,v} = -\frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} p_{x,y}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

onde,

u é a frequência espacial horizontal, para os inteiros $0 \leq u < 8$.

v é a frequência espacial vertical, para os inteiros $0 \leq v < 8$.

$\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{se } u = 0 \\ 1, & \text{caso contrário} \end{cases}$ é um fator de escala de normalização.

$p_{x,y}$ é o valor do pixel nas coordenadas (x, y).

$G_{u,v}$ é o coeficiente DCT nas coordenadas (u, v).

Segue abaixo um exemplo de uma matriz de coeficientes transformados de um bloco de imagem de dimensão 8x8. Os coeficientes indicam que a imagem é formada em grande parte pela imagem base (0,0) da Figura 5, um pouco menos da imagem base (0,1) e os coeficientes de alta frequência pequenos indicam pouco peso das imagens base do canto inferior direito.

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix}$$

Para encerrar essa seção, vale ressaltar que a DCT em certos tipos de imagem diminui os valores dos pixels, uma vez que estes no domínio transformado tendem a ser maiores apenas nas frequências mais baixas. Isso cumpre com o objetivo das transformadas no contexto de imagem introduzido no início da seçâo: (1) reduzir a redundância da imagem, diminuindo o tamanho da maioria dos pixels e (2) identificar as partes menos importantes da imagem, isolando as várias frequências da imagem.

3.2 Run Length Encoding

O Run Length Encoding é um método de compressão de dados que é particularmente eficiente para dados que contêm longas sequências de valores repetidos. O princípio básico do RLE é substituir essas sequências repetidas por um valor único e uma contagem que indica o número de vezes que o valor é repetido.

O algoritmo RLE procura por sequências contínuas de dados idênticos e, para cada sequência de dados repetidos, armazena o valor e a quantidade de repetições. Para reconstruir o dado original, o processo é revertido, onde cada par valor-contagem é expandido para a sequência original. Por exemplo, se tivermos uma sequência de dados como:

$$\text{AAABBBCCDAA}$$

O RLE codifica isso como:

$$3A3B2C1D2A$$

Aqui, $3A$ significa que o caractere A se repete 3 vezes, $3B$ significa que o caractere B se repete 3 vezes, e assim por diante.

A DCT 2D por retornar valores muitos pequenos e muitos zeros após a quantização, é um alvo ideal para o RLE.

3.3 Huffman Encoding

O Huffman Encoding é um método de compressão de dados sem perdas que utiliza uma técnica de codificação baseada na frequência de ocorrência dos símbolos em um conjunto de dados.

A construção da árvore de Huffman começa com a contagem da frequência de cada símbolo (ou caractere) no conjunto de dados. Cada símbolo é representado por

um nó na árvore, onde a frequência do símbolo é o peso do nó. O algoritmo constrói uma árvore binária de Huffman, onde cada nó é um símbolo ou um nó intermediário com a soma das frequências dos seus filhos. Os nós com as menores frequências são combinados primeiro, formando nós intermediários até que todos os símbolos estejam incluídos em uma única árvore.

Após construir a árvore, cada símbolo é codificado com uma sequência de bits que corresponde ao caminho do nó até a raiz da árvore. Símbolos mais frequentes são atribuídos a códigos mais curtos, enquanto símbolos menos frequentes recebem códigos mais longos.

Para decodificar os dados, o processo é revertido. O código binário é percorrido na árvore de Huffman para reconstruir os símbolos originais. Por exemplo, se temos os seguintes símbolos e suas frequências:

A: 45
B: 13
C: 12
D: 16
E: 9
F: 5

A árvore de Huffman pode criar códigos como:

A: 0
B: 101
C: 100
D: 11
E: 000
F: 001

3.4 JPEG

JPEG é um método sofisticado de compressão com perdas ou sem perdas para imagens em cores ou em escala de cinza, que pode combinar os três tópicos citados anteriormente, Huffman Encoding, Run Length Encoding e a DCT. Outros métodos também podem ser utilizados. Os principais passos do JPEG clássico são listados abaixo:

- **Agrupamento em unidades de dados**: Os pixels de cada componente de cor são organizados em grupos de 8×8 pixels, e cada grupo é comprimido separadamente.
- **Transformada discreta de cosseno (DCT)**: Cada unidade de dados é transformada para um mapa de componentes de frequência, o que prepara os dados para a etapa de perda de informação.
- **Quantização**: Cada componente de frequência é dividido por um coeficiente de quantização e arredondado para um inteiro, o que causa perda irreversível de informação.

- **Codificação**: Os coeficientes quantizados são codificados usando uma combinação de codificação de comprimento de execução (RLE) e codificação de Huffman, após a varrimento em zigzag da matriz quantizada.

4 JPEG em SoC

4.1 Objetivo

O objetivo desse projeto será o desenvolvimento de um sistema capaz de realizar a compressão JPEG em um bloco de imagem 8x8. A imagem deve ser transferida para uma memória para que seja processada pelo módulo da DCT descrito em VHDL. A partir da DCT, o Zynq Processing System em C será responsável por realizar os passos seguintes da compressão JPEG. O resultado final será avaliado em termos da precisão dos algoritmos.

4.2 DCT em Hardware

O hardware para a implementação da DCT foi baseado em [CSF77]. Segue na Figura 6 o grafo de fluxo do algoritmo proposto para uma sequência de 8 elementos.

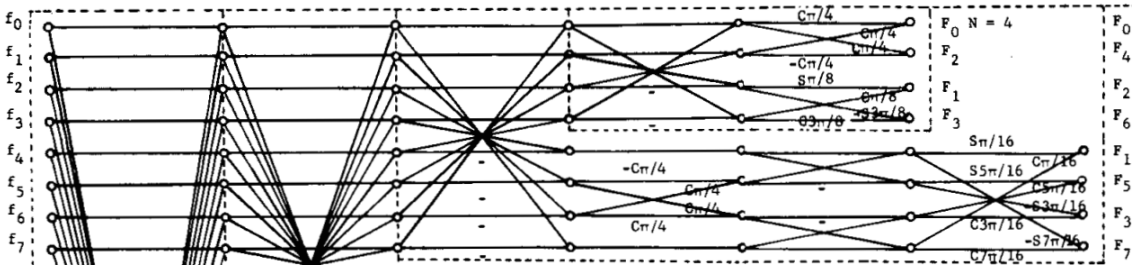

Figura 6: Grafo de fluxo do algoritmo da DCT de 8 pontos em hardware

Para maximizar a velocidade de escrita, foram empregados 8 block memories generators/BRAMs para que a escrita e leitura fosse feita de maneira paralela. Na Tabela 1 tem-se a organização das memórias, onde F_{ij} representa o coeficiente i da sequência de entrada j.

BRAM \ Address	0x00000000	0x00000004	0x00000008	0x0000000C	0x00000010	0x00000014	0x00000018	0x0000001C
$BRAM_0$	F_{00}	F_{01}	F_{02}	F_{03}	F_{04}	F_{05}	F_{06}	F_{07}
$BRAM_1$	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}
$BRAM_2$	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}	F_{26}	F_{27}
$BRAM_3$	F_{30}	F_{31}	F_{32}	F_{33}	F_{34}	F_{35}	F_{36}	F_{37}
$BRAM_4$	F_{40}	F_{41}	F_{42}	F_{43}	F_{44}	F_{45}	F_{46}	F_{47}
$BRAM_5$	F_{50}	F_{51}	F_{52}	F_{53}	F_{54}	F_{55}	F_{56}	F_{57}
$BRAM_6$	F_{60}	F_{61}	F_{62}	F_{63}	F_{64}	F_{65}	F_{66}	F_{67}
$BRAM_7$	F_{70}	F_{71}	F_{72}	F_{73}	F_{74}	F_{75}	F_{76}	F_{77}

Tabela 1: Tabela de BRAMs

A DCT foi implementada em *pipeline*, de forma que cada camada do grafo de fluxo correspondesse a um estágio de cálculo, calculado a cada ciclo de clock.

A DCT em hardware implementada recebe a cada ciclo de clock uma sequência de 8 números, que para o cálculo em uma dimensão, seriam as linhas da matriz de imagem 8x8. Após o calculo da DCT das oito linhas, são calculadas as DCTs para as oito colunas, resultando nos coeficientes finais da DCT-2D do bloco de imagem.

4.2.1 Considerações de tempo

As oito memórias foram empregadas para garantir maior paralelismo ao sistema, tanto na leitura quanto escrita de dados na memória. Caso fosse empregado uma leitura e escrita serial, com a porta de dados com largura de 32 bits, só a leitura e gravação de uma sequência de 8 números demoraria ao menos 8 ciclos de clock para ser realizada, o mesmo que o algoritmo de DCT demora para retornar a sequência de coeficientes de tal entrada. Ou seja um bom tempo é economizado, visto que para uma compressão de imagem, esse processo se repetiria milhares de vezes. Em comparação ao processador ARM, a diferença é ainda maior, visto que o processador demora 3 ciclos de clock para realizar uma soma e até quatro para realizar uma multiplicação. Os valores podem ser encontrados nas tabelas B1 e B5 de https://documentation-service.arm.com/static/5f0370fccafe527e86f5bfb2?token=

4.3 Bock Design

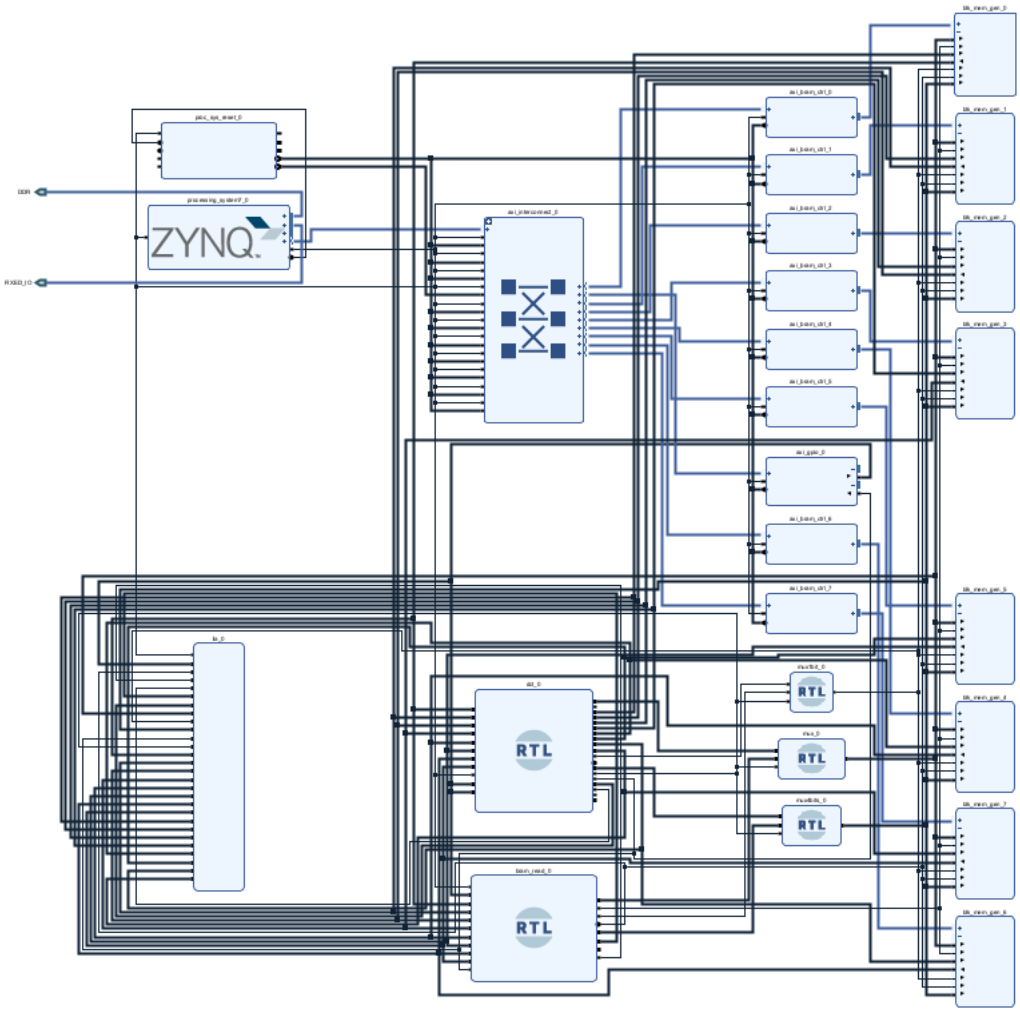

Figura 7: Block Design

4.3.1 DCT

O IP criado no Vivado para a realização da DCT é representado na Figura 8, onde:

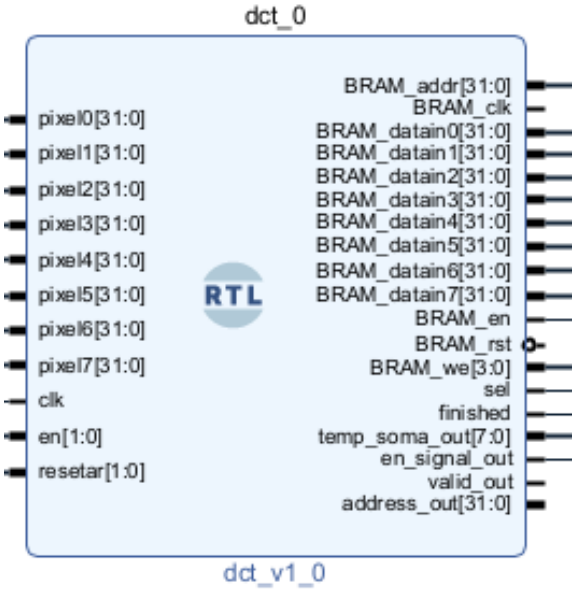

Figura 8: Módulo dct

- **pixel0 a pixel 7**: Sequeência de pixels de entrada.
- **clk**: Sinal de clock.
- **en[1:0]**: [0] Sinal para iniciar a contagem de temp_soma na operação de transformação.
- **resetar[1:0]**: [1] Sinal para resetar o sinal de finished.
- **BRAM_addr[31:0]**: Endereço de gravação na memória.
- **BRAM_datain0[31:0] a BRAM_datain7[31:0]**: Sinal com o valor transformado a ser gravado na BRAM.
- **BRAM_en**: Sinal para habilitar a leitura ou escrita da memória(este último precisa do BRAM_we também).
- **BRAM_we[3:0]**: Sinal para habilitar a escrita na memória.
- **sel**: Sinal de seleção nos multiplexadores. Quando sel=1, a DCT controla a memória, quando qualquer outro valor, bram_read controla as memórias.
- **finished**: Sinal de completude da transformada. É igual a 1 quando completo.
- **Demais**: Sinais de debug. BRAM_clk, BRAM_rst não utilizados.

4.3.2 BRAM Read

O bloco para a leitura de dados nas memórias é apresentado na Figura 9, onde:

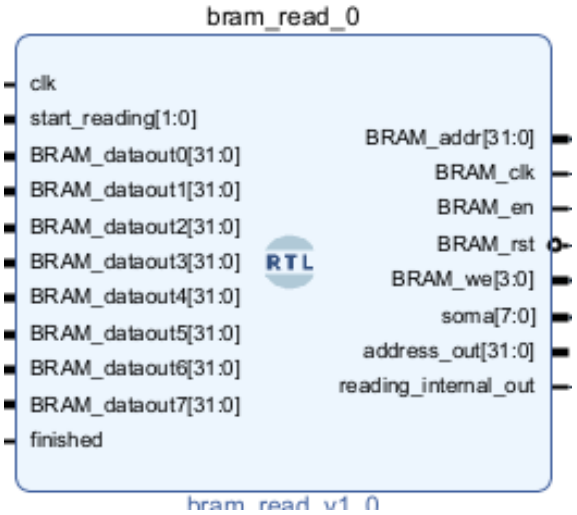

Figura 9: Módulo bram_read

- **clk**: Sinal de clock.
- **start_reading[1:0]**: [0] Sinal para iniciar leitura nas BRAMs.
- **BRAM_dataout0[31:0] a BRAM_dataout7[31:0]**: Sinal de dados lidos das BRAMs.
- **finished**: Sinal de completude da transformada. É igual a 1 quando completo. Usado aqui para restar as variáveis internas do bloco.
- **BRAM_addr[31:0]**: Endereço de leitura na memória.
- **BRAM_clk**: Sinal de clock das BRAMs, o mesmo que clk.
- **BRAM_we[3:0]**: Sinal para habilitar a escrita na memória. Sempre em 0000 aqui.
- **Demais**: Sinais de debug.

4.3.3 Multiplexador de BRAM_addr

O multiplexador para seleção de qual componente controlará o endereçamento nas memórias é mostrado na Figura 10, onde:

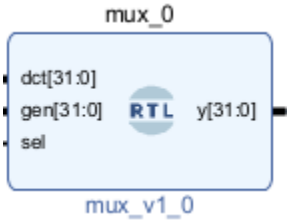

Figura 10: Módulo mux

- **dct[31:0]**: É o sinal de endereço vindo do bloco de DCT.

- **gen[31:0]**: É o sinal de endereço vindo do bloco de leitura das memórias.
- **y[31:0]**: É o sinal de endereço selecionado.
- **sel**: Sinal de seleção. Quando 1, seleciona a dct e quando qualquer outro valor seleciona gen.

4.3.4 Multiplexador de BRAM_en

O multiplexador para seleção de qual componente controlará o BRAM_en nas memórias é mostrado na Figura 11, onde:

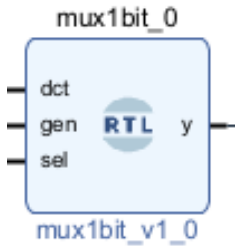

Figura 11: Módulo mux1bit

- **dct**: É o sinal de BRAM_en vindo do bloco de DCT.
- **gen**: É o sinal de BRAM_en vindo do bloco de leitura das memórias.
- **y**: É o sinal de BRAM_en selecionado.
- **sel**: Sinal de seleção. Quando 1, seleciona a dct e quando qualquer outro valor seleciona gen.

4.3.5 Multiplexador de BRAM_we

O multiplexador para seleção de qual componente controlará o BRAM_we nas memórias é mostrado na Figura 12, onde:

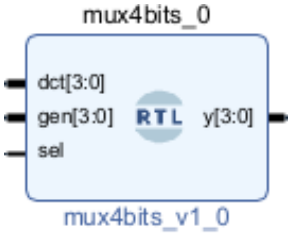

Figura 12: Módulo mux4bits

- **dct[3:0]**: É o sinal de BRAM_wn vindo do bloco de DCT.
- **gen[3:0]**: É o sinal de BRAM_wn vindo do bloco de leitura das memórias.
- **y[3:0]**: É o sinal de BRAM_wn selecionado.

- **sel**: Sinal de seleção. Quando 1, seleciona a dct e quando qualquer outro valor seleciona gen.

4.3.6 Block Design Completo

O block design completo está na última página do documento.

4.4 Simulação e funcionamento do sistema

4.4.1 Organização

O sistema tem como base os módulos bram_read, dct e a PS. As GPIOs são ligadas ao PS pelo intermédio do AXI Interconect, assim como os blocos de memória. Entretanto, os blocos de memória contém um componente adicional para fazer essa comunicação: os controladores de memória. Este sim é conectado ao AXI Interconect. As GPIOs são responsáveis por ler e escrever na PL sinais de controle úteis ao projeto. Como explicado na seção 4.2, são empregados oito blocos de memória concedendo ao projeto maior paralelismo. Não é uma solução ótima, mas é melhor que a leitura e escrita de maneira serial.

4.4.2 Leitura

A simulação para o processo de leitura é mostrado na Figura 13. Sinais em verde são fornecidos pela GPIO, sinais em azul são pertecentes à bram_read, sinais roxos são pertencentes à dct e sinais khaki são pertencentes à BRAM.

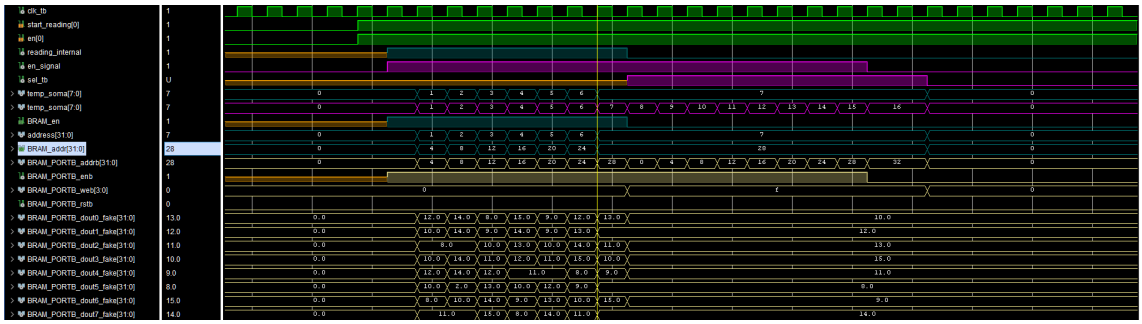

Figura 13: Simulação bram_read

Quando o sinal start_reading fornecido pelo GPIO é igual a B"01", o sinal de reading_internal vai para alto, assim como BRAM_en, quando há uma borda de subida do clock. O sel é iniciado como zero, ou seja, a bram_read que começa controlando as memórias. Quando há uma borda de subida no clk e reading_internal está em alto, a contagem é iniciada. Os endereços de leitura da BRAM aumenta progressivamente de 4 em 4. Os dados de saída da BRAM estão disponíveis no ciclo seguinte de clock. Por exemplo, na primeira borda de subida com reading_internal igual a 1, o endereço a ser lido é o 0, e o valor presente fica disponível para ser

utilizado no ciclo de clock seguinte. O sinal de escrita na BRAM permanece em zero o tempo todo enquanto o sel for igual a zero. Quando temp_soma for igual a 7, a contagem é interrompida e BRAM_en vai a nível baixo. O último dado é lido no endereço 28 em decimal em t=145ns. Os dados lidos são os últimos oito sinais da Figura 13.

4.4.3 DCT e escrita

A simulação com foco no módulo dct é mostrado abaixo:

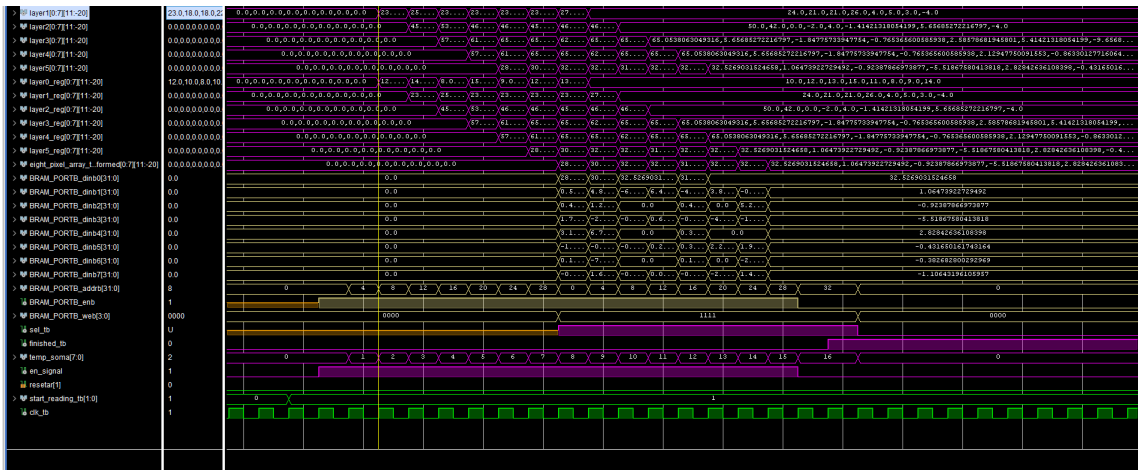

Figura 14: Simulação dct

Os dados lidos pela bram_read, os pixels, são passados para o registrador layer0_reg que após preenchidos são utilizados combinacionalmente pelo sinal layer1, que realiza a primeira camada do algoritmo de DCT. No ciclo seguinte de clock os valores da layer1, são passados ao layer1_reg, cujos valores são utilizados combinacionalmente pelo sinal layer2, que no próximo ciclo de clock é passado ao layer2_reg e assim por diante até eight_pixel_array_transformed. A operação de DCT é concluída ao assim quando layer5_reg é preenchido, entretanto, foi adicionado um registrador a mais para garantir a sincronização do processo de leitura e a DCT. Os cálculos foram feitos utilizando ponto fixo, sendo 12 bits para parte inteira e sinal e 20 bits para a parte fracionária.

O sinal de sel é colocado em nível lógico alto assim que o processo de leitura for concluído, passando o controle das memórias para a DCT. A latência de escrita na memória é de um ciclo de clock. Por exemplo, os valores da Figura 15 terminam de ser gravados no endereço 0 apenas em t=165ns. É por esse motivo que o sinal de finished vai a nível alto só um ciclo depois a contagem de temp_soma atingir o valor de 15. Após o sinal de finished ser disparado, na primeira subida de clock seguinte, todos os sinais de controle interno são resetados. O sinal de finished só é resetado quando a GPIO envia o sinal resetar em nível lógico alto em t=355ns.

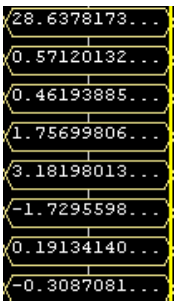

Figura 15: Coeficientes da primeira linha

4.4.4 Software

O software foi desenvolvido no Vitis, o software é rodado no Zynq processig system. O código começa com a configuração dos drivers da GPIO e das BRAMs. Depois é definido a direção de cada bit da GPIO em cada canal. Todos os bits do canal 1 são saídas e todos os bits do canal 2 são entradas. O canal 2 recebe o sinal de finished. Após a configuração é realizado a DCT-2D, realizada pela função DCT_2D em uma matriz 8x8.

A DCT 2D é realizada, primeiramente, gravando-se cada linha da matriz de entrada do software nas memórias. Cada elemento da primeira linha é gravada no endereço zero da sua respectiva memória, a segunda linha no endereço 4 das memórias e assim por diante. Essas linhas são lidas pela DCT em hardware e gravados os coeficientes da transformação na mesma memória. Neste ponto, cada endereço nas memórias corresponde a uma linha da matriz de coeficientes. Por exemplo, o endereço 0 da memória 0 corresponde ao elemento (0,0) da matriz de coeficientes, o endereço 0 da memória 1 corresponde ao elemento (0,1) da matriz de coeficientes e assim por diante. Está feita a DCT-1D. Depois, cada elemento da primeira coluna da DCT-1D é gravada no endereço zero das memórias, a segunda coluna no endereço 4 das memórias e assim por diante. Ao final tem-se a matriz de coeficientes da DCT-2D. Neste ponto, o endereço 0 da memória 0 corresponde ao elemento (0,0) da matriz de coeficientes, o endereço 0 da memória 1 corresponde ao elemento (1,0) da matriz de coeficientes e assim por diante.

O passo seguinte é dividir cada elemento da matriz de coeficientes DCT-2D pelo seu respectivo correspondente na matriz de quantização. A matriz de quantização utilizada foi:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

O resultado das divisões é então arredondado e logo em seguida é aplicado leitura em zigzag.

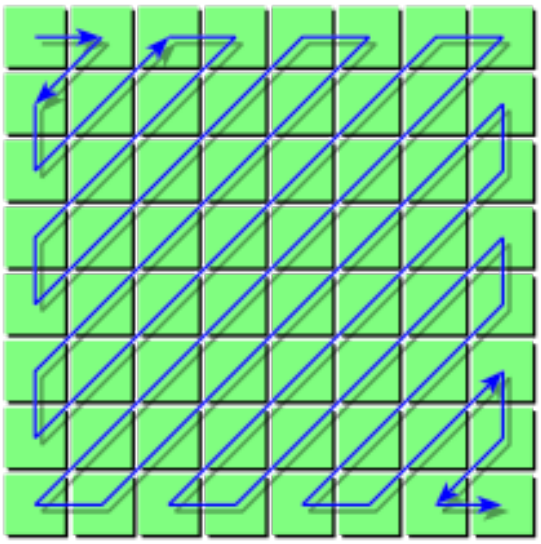

Figura 16: ZigZag

Ao final é aplicado o RLE (*Run Length Encoding*) e o Huffman Encoding, cuja implementação foi adaptada de https://www.programiz.com/dsa/huffman-coding.

5 Resultados

Para comprovar a eficiência e precisão do sistema, foi a aplicado um simples exemplo, contido inclusive na Wikipedia [Wik24]. Foi feita a compressão JPEG para o seguinte bloco de entrada:

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

Os resultados coincidem com os mostrados em [Wik24], salvo por pouca diferença devido a precisão em ponto fixo do sistema desenvolvido.

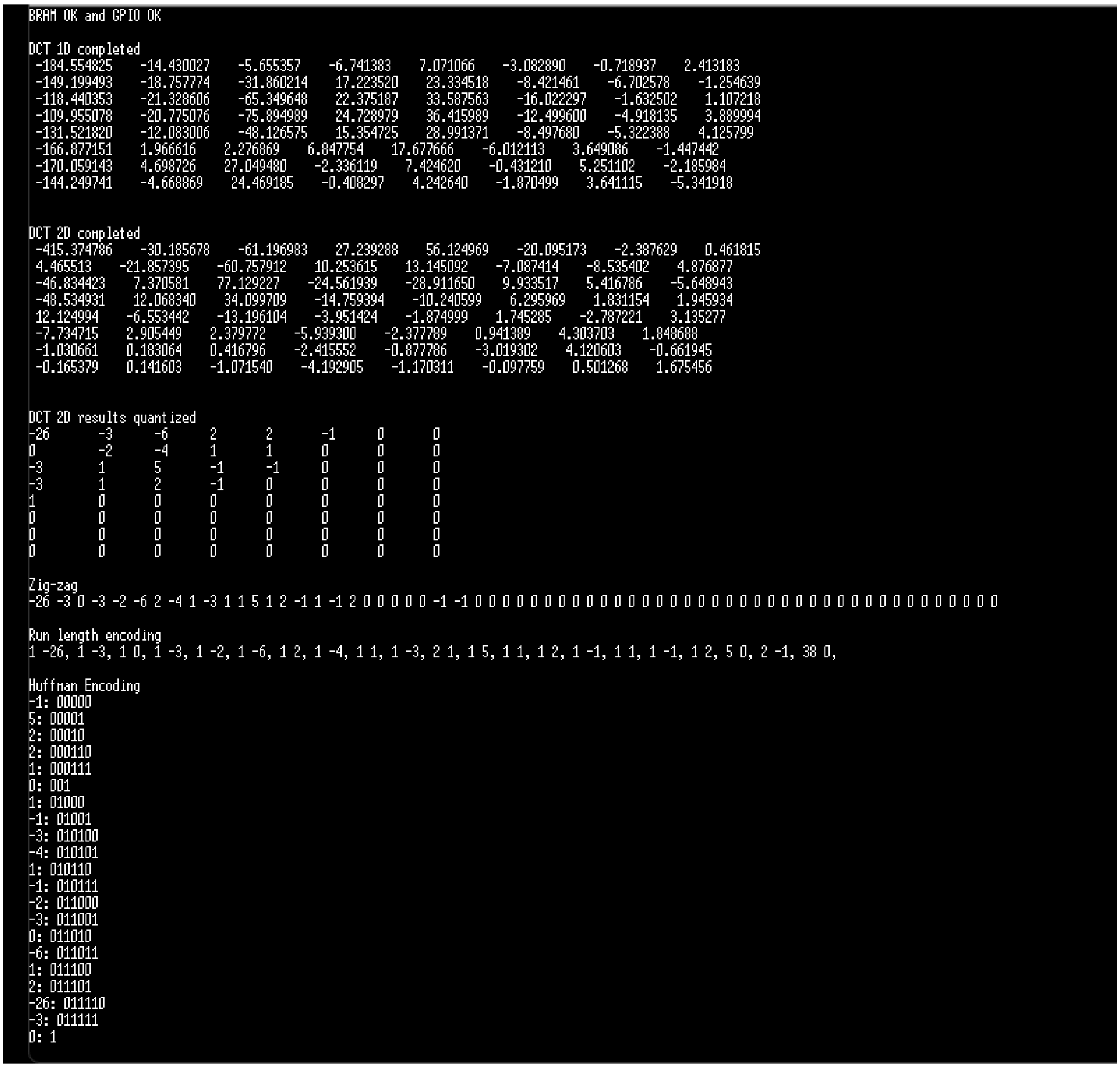

Figura 17: Saída do software no monitor serial

Referências

[CSF77] Wen-Hsiung Chen, C. Smith, and S. Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE Transactions on Communications*, 25(9):1004–1009, 1977.

[Sal10] David Salomon. *Handbook of Data Compression*. Springer, London; New York, 2010.

[Wik24] Wikipedia contributors. Jpeg, 2024. Wikipedia, The Free Encyclopedia.