Implementação de um compressor JPEG em FPGA

Nuno Santos, Pedro Silva e António Esteves
Departamento de Informática, Universidade do Minho, Braga, Portugal
nunosantos@imaginando.net, selu@netcabo.pt, esteves@di.uminho.pt

Resumo

Este documento descreve a implementação com pipeline de um compressor JPEG em FPGA. Para isso, apresenta-se a funcionalidade e implementação dos blocos principais da arquitectura do compressor. O destaque vai para a DCT 2-D, implementada com 2 blocos DCT 1-D e um buffer de transposição. O componente mais crítico em termos de espaço e desempenho é o multiplicador da DCT 1-D, que por ser usado para multiplicar um inteiro por uma de 4 constantes reais possui uma arquitectura baseada em 4 barrel shifters, e é aqui apresentado em maior detalhe.

1 Introdução

O algoritmo JPEG permite comprimir imagens [1]. O formato padrão *baseline* não implica perdas. O processo de compressão de uma imagem no formato JPEG consiste essencialmente de 5 etapas: a conversão do espaço de cores, o processo de *downsampling*, a transformada DCT 2-D, a quantização e a codificação de entropia. Quando se trata de imagens a preto e branco, as duas primeiras etapas não se aplicam [2] [3].

2 O compressor JPEG

2.1 Conversão do espaço de cores

A conversão do espaço de cores é a primeira operação a ser realizada por um compressor JPEG. A razão pela qual se efectua a conversão do espaço de cores é o facto de haver um grau elevado de correlação entre as componentes R, G e B, tornando difícil o processamento de cada uma das informações de cor de forma independente. A imagem é então convertida do espaço de cores RGB para um espaço de cores do tipo luminância e crominância. Existem vários espaços de cores deste tipo. Um deles é o YCbCr, em que a componente Y contém a informação relativa à luminosidade e as componentes Cb e Cr contêm a informação de cor da imagem. Este sistema de cores é o que se utiliza no sistema de televisão PAL, por exemplo.

2.2 *Downsampling*

Como o olho humano consegue ver mais detalhe na componente da luminância do que nas de crominância, pode construir-se um codificador que comprima imagens com maior eficácia, reduzindo a quantidade de bits necessários para representar a informação relativa à crominância. Esta operação é denominada de *downsampling*. Existem várias formas de relacionar as componentes de luminância com as de crominância na implementação do *downsampling*, dependendo da aplicação. Essa relação é geralmente especificada na forma X:Y:Z. Por exemplo, 4:1:1 significa que para cada 4 componentes Y, existe apenas uma componente de crominância. Suponhamos que temos uma fracção de quatro pixeis de uma imagem colorida qualquer, onde cada componente de cor utilize 8 bits, são necessários 32 bits para representar cada uma das três informações de cor destes quatro pixeis e, no total, são utilizados 96 bits para representar esta fracção da imagem. Com a operação de *downsampling* na taxa de 4:1:1, a informação de luminância continua a utilizar 32 bits (4x8), mas as informações de crominância passam a utilizar 8 bits cada (1x8), formando um total de 48 bits, que é metade dos 96 iniciais, havendo uma taxa de compressão de 50%.

2.3 DCT 2-D

A transformada discreta do co-seno em duas dimensões é utilizada para transformar a representação da informação do domínio espacial para o domínio das frequências. Após a transformação de domínio, as frequências mais elevadas, que tendem a contribuir menos para a imagem, são atenuadas ou mesmo eliminadas pelo processo de quantização. Nesta fase, cada componente da imagem (Y,Cb,Cr) é organizada em pequenos blocos de 8x8 pixeis. Cada bloco é posteriormente convertido para o espaço das frequências usando a transformada discreta do co-seno de duas dimensões.

2.4 Quantização

O olho humano é bom a distinguir pequenas diferenças na luminosidade sobre uma área relativamente grande mas não é tão bom a distinguir a amplitude exacta de uma frequência alta de variação de luminosidade. Este facto permite reduzir de forma significativa a informação armazenada relativa às componentes de frequência alta. Isto é feito dividindo cada componente no domínio das frequências por uma constante para essa componente e, de seguida, arredondando para o inteiro mais próximo. Esta é a operação que introduz mais perdas no processo de compressão.

2.5 Codificação de entropia

Após o processo de quantização, a matriz resultante terá muitas ocorrências de zeros. Os valores diferentes de zero têm maior probabilidade de estarem concentrados no canto superior esquerdo da matriz de coeficientes, pois a DCT 2-D tende a concentrar a maior parte da energia da imagem no canto superior esquerdo da matriz. Por esta razão, esta matriz é lida em ziguezague, para que as ocorrências de zeros apareçam em grandes sequências, potenciando as técnicas de compressão utilizadas pelo codificador de entropia. A codificação de entropia baseia-se em três técnicas de compressão: *Variable Length Coding* (VLC), *Run-Length Encoding* (RLE) e codificação de *Huffman*.

3 Implementação da DCT 2-D

A DCT 2-D pode ser implementada tirando proveito da propriedade da separabilidade. Deste modo, a arquitectura da DCT 2-D utiliza dois blocos DCT 1-D e um *buffer* de transposição, que recebe os resultados do primeiro bloco DCT 1-D, armazena-os linha a linha e entrega-os coluna a coluna ao segundo bloco DCT 1-D (figura 1). Assim, a saída da DCT 2-D fica organizada coluna a coluna, ou seja, as primeiras oito saídas formam a coluna 0 da matriz dos resultados. Os dois blocos DCT 1-D necessários ao cálculo da DCT 2-D poderiam ser implementados com um único bloco, usando realimentação, mas ao usar dois blocos em *pipeline* aumenta-se o desempenho.

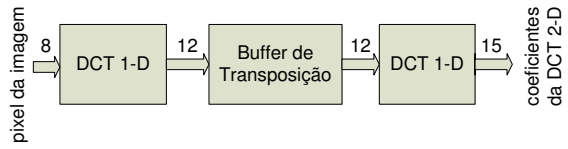

Figura 1: Arquitectura da DCT 2-D.

Antes de ser processado o cálculo da DCT 2-D, o pixel de entrada deve passar por um deslocamento de nível. Este deslocamento faz com que a média dos valores de entrada passe de 128 para 0, possibilitando uma maior uniformidade do processamento da imagem. Esta operação resume-se a uma simples subtracção de 128 a todos os valores de entrada, que passam então, a estar contidos no intervalo de valores de -128 a 127. Esta operação pode ser simplesmente implementada através da inversão do oitavo bit.

3.1 DCT 1-D

A DCT 1-D pode ser implementada através de seis estágios, possibilitando novamente a utilização de *pipeline*, aumentando assim o desempenho do algoritmo. Em cinco dos seis estágios efectua-se somas ou subtracções e no restante executam-se multiplicações. As várias operações de cada estágio partilham um único operador aritmético. A DCT pode ser descrita pelo seguinte algoritmo com 6 estágios [4]:

Estágio 1: *b0=a0+a7, b1=a1+a6, b2=a3-a4, b3=a1-a6, b4=a2+a5, b5=a3+a4, b6=a2-a5, b7=a0-a7.*

Estágio 2: *c0=b0+b5, c1=b1-b4, c2=b2+b6, c3=b1+b4, c4=b0-b5, c5=b3+b7, c6=b3+b6, c7=b7.*

Estágio 3: *d0=c0+c3, d1=c0-c3, d2=c2, d3=c1+c4, d4=c2-c5, d5=c4, d6=c5, d7=c6, d8=c7.*

Estágio 4: *e0=d0, e1=d1, e2=m3*d2, e3=m1*d7, e4=m4*d6, e5=d5, e6=m1*d3, e7=m2*d4, e8=d8.*

Estágio 5: *f0=e0, f1=e1, f2=e5+e6, f3=e5-e6, f4=e8+e3, f5=e8-e3, f6=e2+e7, f7=e4+e7.*

Estágio 6: *S0=f0, S1=f4+f7, S2=f2, S3=f5-f6, S4=f1, S5=f5+f6, S6=f3, S7=f4-f7.*

em que $m1 = cos(4\pi/16)$, $m2 = cos(6\pi/16)$, $m3 = cos(2\pi/16) - cos(6\pi/16)$ e $m4 = cos(2\pi/16) + cos(6\pi/16)$.

A arquitectura correspondente a este algoritmo da DCT 1-D encontra-se na figura 2. A arquitectura é composta de 6 níveis de pipeline, cada um com uma unidade aritmética (somador/subtractor ou multiplicador), multiplexadores para seleccionar os operandos e um registo duplo, que permite um desempenho maior.

Numa fase inicial, os somadores a utilizar nos cinco estágios que envolvem somas ou subtracções e no multiplicador, são do tipo *ripple carry*. Estes somadores são simples de projectar, ocupam uma área reduzida, mas apresentam um desempenho baixo. Para efectuar somas e subtracções (entre $op1$ e $op2$) com o mesmo *hardware*, só é necessário usar um multiplexador para seleccionar $op2$ ou $/op2$ e incluir um sinal de $carryIn$, que assume o valor 0 ou 1, consoante se trata de soma ou subtracção.

O multiplicador necessário ao estágio 4 da primeira (segunda) DCT 1-D efectua o produto $a * m$, em que a é um inteiro de 11 (15) bits e m é uma constante real $\in \{m1, m2, m3, m4\}$. Efectuando algumas simplificações, cada constante m_i pode ser aproximada por um valor real com 1 bit para a parte inteira, 9 bits para a parte fraccionária e conter apenas 4 bits a '1' (tabela 1). De acordo com esta aproximação, a multiplicação $a * m$ pode ser calculada da seguinte forma:

$$\begin{aligned} a*m &= a*(2^{-i}+2^{-j}+2^{-k}+2^{-l}) \text{ com } -i,-j,-k,-l \leq 0 \\ &= a*(\tfrac{1}{2^i}+\tfrac{1}{2^j}+\tfrac{1}{2^k}+\tfrac{1}{2^l}) \\ &= (\tfrac{a}{2^i}+\tfrac{a}{2^j}+\tfrac{a}{2^k}+\tfrac{a}{2^l}) \\ &= (a \gg i)+(a \gg j)+(a \gg k)+(a \gg l) \end{aligned}$$

em que a operação $(a \gg i)$ denota o deslocamento aritmético, em i posições, do valor a representado em complemento para 2.

constante	valor decimal	valor binário aproximado	bits a '1' (-i, -j, -k, -l)
m1	0,707107	0,101101000	-1, -3, -4, -6
m2	0,382683	0,011000101	-2, -3, -7, -9
m3	0,541196	0,100010101	-1, -5, -7, -9
m4	1,306563	1,010011000	0, -2, -5, -6

Tabela 1: Constantes da multiplicação da DCT 1-D.

De acordo com a expressão anterior, a multiplicação $a * m$, em que $m \in \{m1, m2, m3, m4\}$,

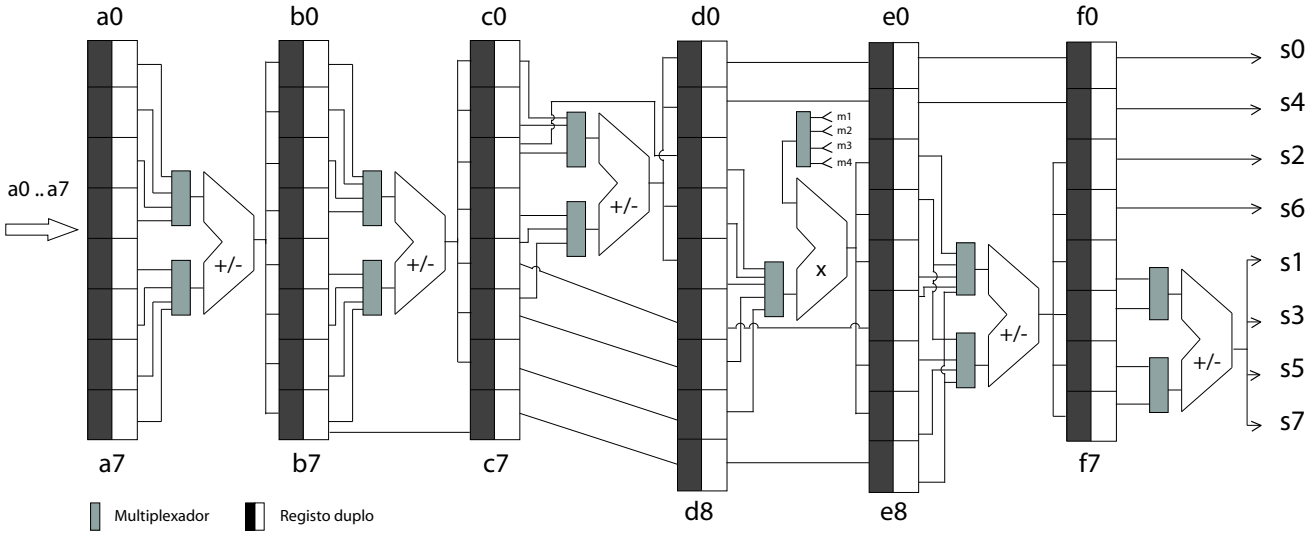

Figura 2: Arquitectura da DCT 1-D.

pode ser implementada recorrendo a 4 *barrel shifters* (BS_i, BS_j, BS_k, BS_l) e 3 somadores (figura 3). Reduzindo o número de deslocamentos ao mínimo, o *barrel shifter* i (BS_i) faz deslocamentos de {0,1,2} posições, BS_j faz deslocamentos de {0,1,3} posições, BS_k faz deslocamentos de {0,1,3} posições e BS_l faz deslocamentos de {0,3} posições (tabela 2).

	BS_i	BS_j	BS_k	BS_l
a*m1 =	$(a \gg 1) +$	$(a \gg 3) +$	$(a \gg 4) +$	$(a \gg 6)$
a*m2 =	$(a \gg 2) +$	$(a \gg 3) +$	$(a \gg 7) +$	$(a \gg 9)$
a*m3 =	$(a \gg 1) +$	$(a \gg 5) +$	$(a \gg 7) +$	$(a \gg 9)$
a*m4 =	$(a \gg 0) +$	$(a \gg 2) +$	$(a \gg 5) +$	$(a \gg 6)$
shifts a realizar	0,1,2	0,1,3	0,1,3	0,3

Tabela 2: Valor dos deslocamentos a efectuar pelos 4 *barrel shifters* $BS_{i,j,k,l}$ do multiplicador.

A arquitectura resultante para o multiplicador da primeira/segunda DCT 1-D é ilustrada na figura 3. O valor dos sinais $shift\_BS_{i,j,k,l}$ que controlam o número de deslocamentos a efectuar por cada *barrel shifter* $BS_{i,j,k,l}$, de modo a obter-se a multiplicação de a por $m_{1,2,3,4}$ é apresentado na tabela 3.

	a*m1	**a*m2**	**a*m3**	**a*m4**
$shift\_BS_i$	01	10	01	00
$shift\_BS_j$	01	01	11	00
$shift\_BS_k$	00	11	11	01
$shift\_BS_l$	00	11	11	00

Tabela 3: Sinais que controlam o número de deslocamentos dos *barrel shifters* do multiplicador.

As razões porque se escolheu o *barrel shifter*, um componente combinacional, em vez do *shift register* sequencial, para efectuar os deslocamentos necessários ao multiplicador da DCT 1-D, foram (i) o número de deslocamentos ser variável (entre 0 e 3 posições) e (ii) o *barrel shifter* permitir maior desempenho, ao não exigir n ciclos de relógio para efectuar um deslocamento de n posições. Uma possível arquitectura do *barrel shifter* que efectua deslocamentos de $shift_{1:0} (= 0..3)$ posições sobre um valor de n bits em complemento para 2, consiste numa estrutura com 2 níveis de multiplexadores 2:1 (figura 4).

3.2 *Buffer* de transposição

O *buffer* de transposição é utilizado para ligar os dois blocos DCT 1-D, em que o resultado do primeiro DCT 1-D é armazenado linha a linha, numa memória de 64 palavras de 12 bits, e lido pelo segundo DCT 1-D coluna a coluna, a partir duma memória também com 64 palavras de 12 bits.

A implementação do *buffer* de transposição poderia utilizar pequenos blocos de memória RAM de porta simples ou registos. Como o objectivo é implementar o compressor em FPGA, a preferência por blocos de memória justifica-se pelo facto de nestes dispositivos a memória interna distribuída ser um recurso menos crítico do que os registos. Este *buffer* é implementado com duas memórias RAM, três multiplexadores e um bloco de controlo. Para obter uma implementação optimizada numa FPGA Virtex-4, a descrição das memórias foi feita instanciando componentes RAM64X1S [5]. Cada componente RAM64X1S é uma memória SRAM com 64 palavras de 1 bit. As duas memórias operam de forma intercalada: enquanto uma é usada para a escrita a outra é usada para leitura. Considerando as duas memórias preenchidas, a memória usada para escrita fica completamente preenchida pelos dados gerados pela primeira DCT 1-D, exactamente no mesmo momento em que foi consumido, pela segunda DCT 1-D, todo o conteúdo de memória usada para leitura. No próximo ciclo de relógio, inverte-se a situação, passando a ser usada para escrita a memória que antes estava a ser usada para leitura, e vice-versa.

4 Conclusões

O trabalho apresentado, em execução por parte de 2 alunos de licenciatura, encontra-se em implementação,

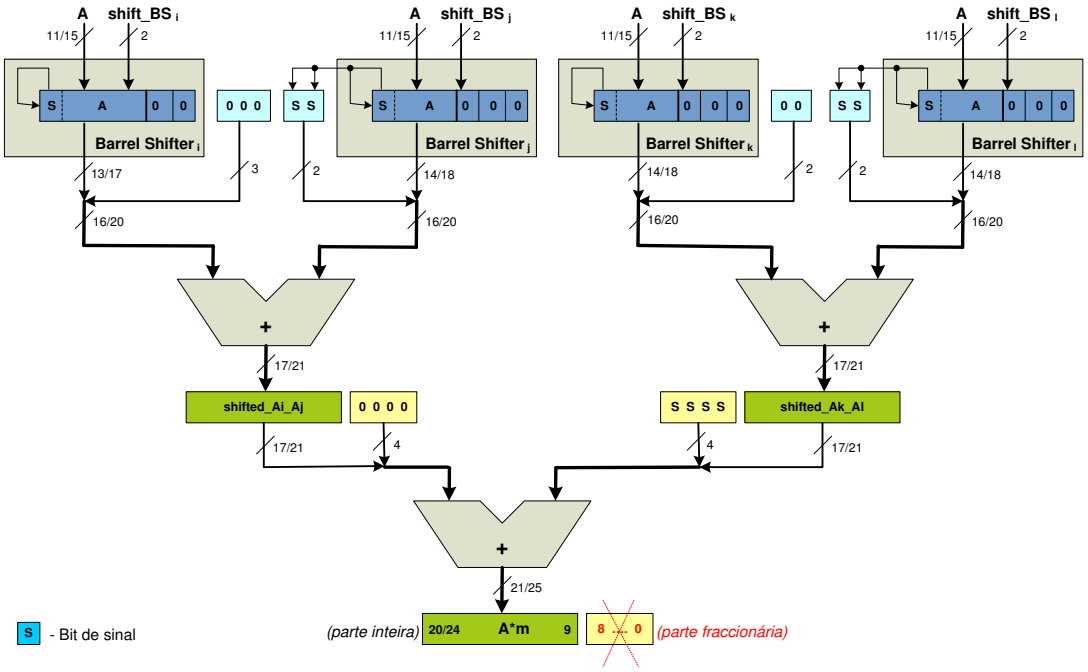

Figura 3: Arquitectura do multiplicador a usar nas DCT 1-D.

tendo ainda poucos resultados práticos. De momento foi escolhida, avaliada e descrita em VHDL a arquitectura do bloco DCT 2-D. Falta concluir a máquina de estados correspondente à unidade controlo que vai gerir o funcionamento em pipeline das duas DCT 1-D e do *buffer* de transposição. Nas próximas etapas vai simular-se a descrição VHDL e obter-se uma implementação para uma FPGA Virtex4. Com o trabalho realizado, os alunos já atingiram os principais objectivos pedagógicos e de investigação idealizados para este trabalho, ou seja, a motivação e aprendizagem das problemáticas de síntese de sistemas com elevado desempenho (pipeline) ao menor custo (espaço) e a sua implementação com lógica reconfigurável.

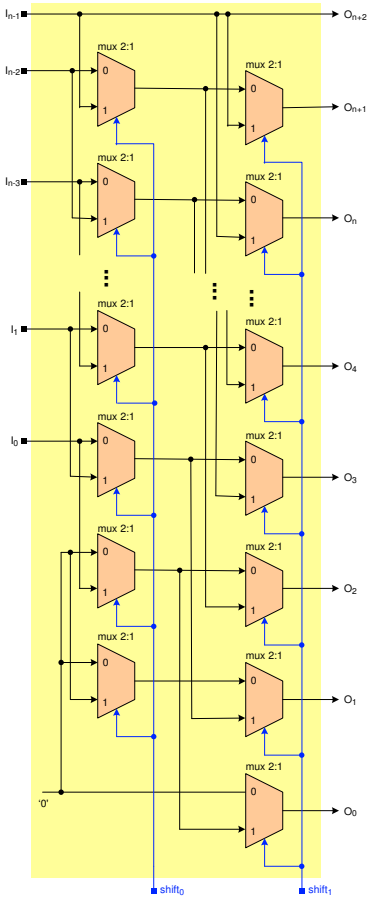

Figura 4: *Barrel shifter* utilizado no multiplicador da DCT 1-D, para deslocamentos entre 0 e 3 posições.

Referências

[1] Gregory Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, (Vol 34, N 4):30–44, April 1991.

[2] Luciano Volcan Agostinin. Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG. Master's thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil, March 2002.

[3] Luciano V. Agostini, Ivan S. Silva, and Sergio Bampi. Pipelined Fast 2-D DCT Architecture for JPEG Image Compressions. In *Proceedings of the 14th Symposium on Integrated Circuits and Systems Design*, September 2001. Brasil.

[4] Mario Kovac and N. Ranganathan. JAGUAR: A Fully Pipelined VLSI Architecture for JPEG Image Compression Standard. *Proceedings of the IEEE*, (Vol 83, N 2):247–258, February 1995.

[5] Xilinx. Virtex-4 Libraries Guide for HDL Designs (ISE 8.1i), 2005.