IMPLEMENTAÇÃO DA DCT 2D EM ARQUITETURAS RECONFIGURÁVEIS UTILIZANDO A X4CP32

Arnaldo Azevedo, Rodrigo Soares, Ivan Saraiva Silva

Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
[arnaldo, rodsoares]@lcc.ufrn.br, ivan@dimap.ufrn.br

ABSTRACT

The X4CP32 is a coarse grain reconfigurable architecture, that consists of two hierachic levels of abstraction: grain and cell. Each one with theirs own mecanisms of progamability and configurability. In this paper will be present a Two Dimensional Discrete Cossine Tranform (2D DCT) implementation in a reconfigurable architecture. A architecture with 40 grains has a 141,1 monochromatic images (640x480) per second. The results is comparated with a ASIC and a software implementation.

RESUMO

A X4CP32 é uma arquitetura reconfigurável de granularidade grossa, desenvolvida a partir de dois níveis hierárquicos de abstração: o grão e a célula. Cada nível apresentando mecanismos próprios de programação e configuração. Neste artigo apresenta-se a implementação da Transformada Discreta do Cosseno em duas dimensões (DCT 2D) em uma arquitetura reconfigurável. Com a utilização de uma arquitetura constituída de 40 grãos foi possível obter um desempenho de 141,4 imagens monocromáticas (640x480) por segundo. Os resultados obtidos foram comparados com os resultados de um ASIC e uma implementação software.

IMPLEMENTAÇÃO DA DCT 2D EM ARQUITETURAS RECONFIGURÁVEIS UTILIZANDO O X4CP32

Arnaldo Azevedo, Rodrigo Soares, Ivan Saraiva Silva

Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
[arnaldo, rodsoares]@lcc.ufrn.br, ivan@dimap.ufrn.br

RESUMO

A X4CP32 é uma arquitetura reconfigurável de granularidade grossa, desenvolvida a partir de dois níveis hierárquicos de abstração: o grão e a célula. Cada nível apresentando mecanismos próprios de programação e configuração. Neste artigo apresenta-se a implementação da Transformada Discreta do Cosseno em duas dimensões (DCT 2D) em uma arquitetura reconfigurável. Com a utilização de uma arquitetura constituída de 40 grãos foi possível obter um desempenho de 141,4 imagens monocromáticas (640x480) por segundo. Os resultados obtidos foram comparados com os resultados de um ASIC e uma implementação software.

1. INTRODUÇÃO

O avanço da tecnologia de telecomunicação tornou realidade aplicações que usam vídeo digital, tais como teleconferência e transmissões multimídia. Isso não seria possível sem mecanismos que reduzissem a banda passante necessária à transmissão e o espaço destinado ao armazenamento dessas mídias. Diversos padrões foram implementados com o fim de compactar essas mídias sem que se tenham grandes perdas. Os padrões JPEG (para imagens) e MPEG (para vídeos) se tornaram populares nessa área e são utilizados numa grande gama de aplicações, sendo os padrões escolhidos para os DVD's e para a TV de alta resolução (HDTV).

Para atingir altas taxas de compressão esses padrões utilizam-se, entre outras técnicas, da Transformada Discreta do Cosseno em duas dimensões (DCT 2D), que transforma uma representação da mídia no domínio espacial para uma representação no domínio da freqüência.

Esse artigo visa demonstrar a flexibilidade e o poder das Arquiteturas Reconfiguráveis (AR) [1, 2, 3, 4, 5] em aplicações que requerem grande fluxo de dados e poder de processamento.

Na próxima sessão será apresentada a AR escolhida para a implementação da DCT 2D. Na terceira sessão serão discutidos os detalhes da implementação. Os resultados obtidos e as comparações com outras implementações serão apresentados na sessão 4. Por fim, as conclusões do artigo são apresentadas na sessão 5.

2 O X4CP32

Nesta sessão será apresentada a arquitetura reconfigurável X4CP32 [6], na qual foi realizada a implementação da Transformada Bidimensional do Cosseno.

2.1 A Arquitetura

O X4CP32 é composto por grãos e células, bem como por uma unidade de controle responsável pelo controle do fluxo de configuração e execução em toda a arquitetura.

2.1.1. O Grão

O grão é a unidade que fornece a possibilidade de reconfiguração da arquitetura. Esta unidade consiste de 4 células, uma memória de 2 leituras e 1 escrita com 64k posições de 32 bits (G-MEM), além de um controle interno. O grão está ligado a seus 4 vizinhos (norte, sul, leste e oeste), através da interface de suas células, e a 3 barramentos, que serão detalhados mais adiante neste artigo. Através de dois desses barramentos o grão pode se comunicar com qualquer outro grão em no máximo 2 etapas. Pelo terceiro barramento o grão recebe dados da unidade de controle. Os barramentos dedicados que fazem a interface entre células de grãos vizinhos são half-duplex.

O grão pode assumir dois *modos de execução* na arquitetura: processador ou bloco de ULA's, que serão detalhados mais adiante neste artigo.

2.1.2. A Célula

A célula é um microprocessador, capaz de executar operações aritméticas e lógicas e manter o fluxo de

execução de um programa. É a unidade responsável pela computação em si. Todas as células são compostas, como demostrada na **Figura 1** por uma ULA (ALU), uma memória interna (C-MEM) de 1024 posições (32 bits), uma pilha (C-LIFO) de 64 posições, um registrador visível ao programador (ACC), uma lógica de controle interno (CONTROLE), seis portas de comunicação (N, S, W, E, M e D) e um caminho de dados dedicado a roteamento.

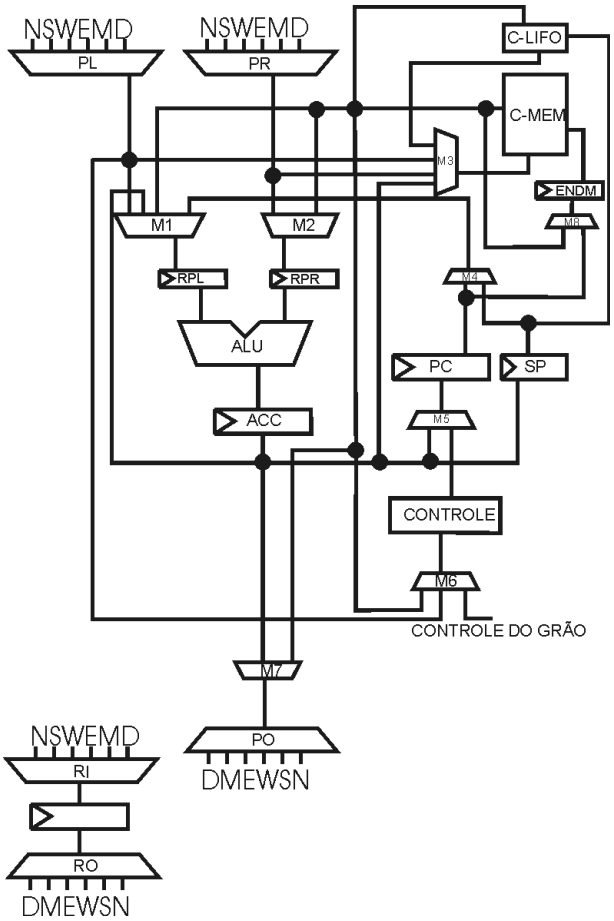

Figura 1. Arquitetura da Célula

A ULA opera com 32 bits e faz operações aritméticas com operandos inteiros e de ponto-flutuante, bem como operações lógicas. Seus operandos podem vir da C-MEM, de uma porta de comunicação, ou do registrador acumulador (ACC).

Das seis portas de comunicação disponíveis na célula, 5 são utilizadas para comunicação com as células vizinhas. A **Figura 2** mostra a ligação das células com suas vizinhas, não sendo apresentada a ligação com a G-MEM. Três dessas ligações são para células do mesmo grão, as outras 2 ligam a célula às células de grãos vizinhos. A sexta porta destina-se à comunicação entre a célula e a memória interna do grão, G-MEM.

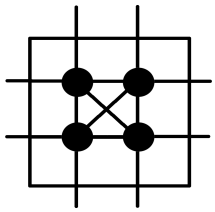

Figura 2. Ligações entre Células

2.1.3. O Controle Principal

A entidade aqui referenciada como *Controle Principal* é aquela que faz o papel de controlador nos processadores convencionais. Ela é responsável pela leitura das instruções e dados da memória externa, pela configuração de todos os grãos e pela alimentação dos grãos com dados, através de um barramento compartilhado por todos os grãos.

2.2. Comunicações

A X4CP32 conta com diversos níveis de comunicação. O primeiro deles é a comunicação entre células do mesmo grão. Essa comunicação é feita através de barramentos dedicados half-duplex de 32 bits. Cada célula em um grão está ligada a todas as outras através desses barramentos.

A comunicação entre grãos vizinhos, a partir de células adjacentes, é feita com o mesmo custo que a comunicação entre as células internas de um grão. As células de um grão dedicam duas de suas portas de comunicação a comunicação com os grãos vizinhos. Cada grão tem, portanto 2 portas de comunicação com cada um de seus 4 vizinhos.

A comunicação entre grãos distantes pode ser feita de duas formas: por *roteamento* ou por *barramento*. Todas as células podem ter suas portas (ligações com células vizinhas) ativadas ou desativadas. Essas portas podem servir para entrada, saída ou para roteamento. Quando uma porta de uma célula é destinada a roteamento, o dado que chega por ela é simplesmente repassado para a porta de saída (de roteamento). Dessa forma é possível criar rotas para o fluxo de dados sem que isto implique em custo de processamento.

A outra forma de comunicação de um grão com grãos distantes se faz por meio de barramentos. Existem barramentos horizontais interligando todos os grãos de uma mesma linha e barramentos verticais interligando todos os grãos de uma mesma coluna.

2.3. Instruções

O X4CP32 conta com 2 níveis de instruções, as *Instruções de Grão* e as *Instruções de Célula*. Como o próprio nome indica, as instruções de grão são aquelas enviadas do Controle Principal para um grão, e as instruções de célula são aquelas normalmente enviadas de um grão para suas células. Existem 6 instruções de grão e 32 instruções de célula[6].

4.1. Instruções de Grão

As *Instruções de Grão* são responsáveis pela troca de dados com o Controle Principal, pela troca de dados pelo barramento entre grãos distantes e pela configuração interna do grão. O Controle Principal envia instruções para os grãos fazendo uso dos barramentos horizontais e verticais. O controle do grão então lê e decodifica as instruções, executando-as ou dividindo-as em um conjunto de instruções de célula, conforme a necessidade.

4.2. Instruções de Célula

As *Instruções de Célula* são, na maioria das vezes, enviadas do controle do grão para as células, mas podem também ser enviadas de uma célula para outra, quando o grão estiver configurado no *Modo de Execução Processador* (detalhes mais adiante neste artigo).

O conjunto de instruções é formado por 32 instruções, das quais 3 são para a configuração da célula e as demais destinam-se a operações aritméticas, lógicas e controle do fluxo de programa.

2.4. Configuração do Grão

O grão, por ser a maior unidade configurável do X4CP32, pode assumir diversas configurações. Essas configurações são chamadas *Modos de Execução*. Os *Modos de Execução* definem o comportamento do conjunto de células de um grão.

Existem dois *Modos de Execução* na X4CP32: Processador e Bloco de ULAs.

Para a configuração do grão no Mode de Execução Processador o Controle Principal envia a instrução de configuração e em seguida o código a ser executado.

Para a configuração em Bloco de ULA o Controle Principal envia a intrução de configuração contendo as operações a serem realizadas em cada Célula e em seguida uma intruções de grão para que sejam configuradas as portas de entrada e de saída de duas células. Em seguida é enviada uma nova instrução de grão para a configuração das duas Células restantes.

2.4.1. Processador

No modo de execução *Processador* o grão une suas 4 células para constituir um processador com uma célula assumindo a unidade de controle de fluxo, chamada célula PC, e as demais agindo como suas ULA's.

Em cada Instrução de Célula há um campo de 2 bits com o endereço da célula-destino. É com base neste endereço que a célula PC irá enviar instruções para as outras células de seu grão. O fluxo de execução do programa continua enquanto houverem instruções a serem executadas, ou até que o grão receba uma outra instrução de configuração. O registrador PC da célula age como o registrador PC de um microprocessador comum, apontando a próxima instrução (posição de C-MEM) a ser executada.

2.4.2. Bloco de ULAs

No modo de execução *Bloco de ULAs* o grão age como um array de células, ou seja, como um conjunto interconectado de células independentes. O conjunto de células do grão recebe um dado por uma entrada particular, realiza a operação para a qual foi configurado e o envia para uma de suas saídas, de acordo com a conexão definida.

2.5 Sincronismo

O sincronismo é baseado na dependência de dados e implementado usando-se *flags* enviado/recebido. Quando um resultado é enviado para uma célula adjacente a célula remetente configura o bit do flag como enviado, a célula destino fica lendo esse bit e quando está ativo ela lê o dado e configura o bit como recebido.

3. A IMPLEMENTAÇÃO DA DCT 2D

A Transformada Discreta do Cosseno (DCT) em duas dimensões é utilizada para se transformar uma representação da informação no domínio do espaço em uma representação no domínio da freqüência. Esse procedimento é utilizado na codificação de imagens no formato JPEG.

O cálculo da DCT em duas dimensões para a codificação JPEG se dá a partir de matrizes de 8x8 pixels. É sobre essas matrizes que essa implementação foi baseada.

3.1. O Algoritmo Utilizado

Devido ao alto grau de complexidade do cálculo real da transformada discreta do cosseno em duas dimensões, apresentada na **Figura 3**, suas implementações em hardware são baseadas em simplificações e aproximações.

$$X(u,v) = \frac{2}{N} C(u)C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i,j) \cos\left(\frac{(2i+1)u\pi}{2N}\right) \cos\left(\frac{(2j+1)v\pi}{2N}\right)$$

where $C(0) = 1/\sqrt{2}$, $C(u) = C(v) = 1$ for $u, v \neq 0$

Figura 3. DCT 2D real

O mecanismo utilizado neste trabalho faz uso da propriedade da separabilidade da DCT 2D. Sendo assim, o cálculo pode ser dividido em duas transformadas unidimensionais. Na primeira delas são calculadas as linhas da matriz de entrada, na segunda são calculadas as colunas da matriz de coeficientes, gerada pela primeira DCT 1D. Isso reduz significativamente a complexidade do cálculo da transformada, sendo por isso bastante utilizado em implementações em hardware. Neste caso a área ocupada também obtém uma diminuição expressiva. Uma representação genérica da DCT bidimensional através de duas DCT unidimensionais está exposta na **Figura 4**

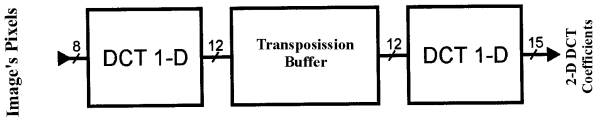

Figura 4. Arquitetura genérica da DCT 2D

A utilização deste método reduz o problema ao cálculo da DCT unidimensional com oito entradas. Para isso foi escolhido o algoritmo proposto por [7] e [8], que é exibido na **Figura 5**. A escolha tem como base a quantidade de operações necessárias para a computação, sendo utilizadas 5 multiplicações e 29 adições neste algoritmo, e a independência entre seus passos, o que torna possível a implementação um pipeline.

3.2. A Implementação da DCT Unidimensional

A partir do algoritmo acima foi construída uma árvore de fluxo dos dados e esta arvore foi implementada no X4CP32. O resultado da transformação manual da árvore em um fluxo no processador pode ser visto na **Figura 6**.

Como se pode observar na **Figura 6,** a implementação manual ocupa uma área de 4x5 grãos, sendo os grãos com fundo cinza configurados como *Processadores* e os demais como *Bloco de ULA*.

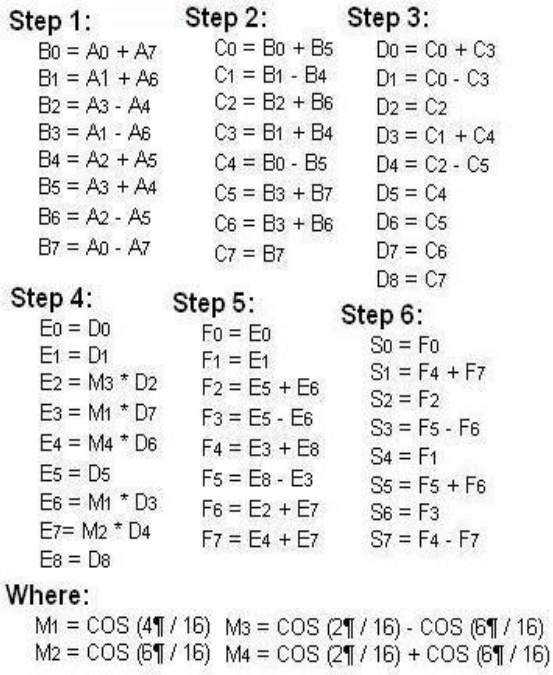

Figura 5. Algoritmo da DCT 1D

As setas com pontas em "V" representam uma saída de resultado que são enviados para um dos grãos processadores. Os roteamentos estão representados por uma linha de ligação entre a seta de entrada e a de saída, lembrando que os dados que servem de entrada para roteamento podem ser utilizados simultaneamente para as operações normais da Célula. As transmissões de dados via barramento estão representadas por indicações na forma M(XX).

As operações grafadas como "~" representam uma solução para o problema de um determinado dado precisar ser roteado para duas Células distintas. Essa solução consiste em fazer uma configuração de roteamento normal entre a entrada e uma das saídas desejadas e utilizar a mesma entrada como operador de uma passagem pela ULA, ou seja, fazer com que a saída de ULA seja o próprio dado de entrada, e este resultado direcionado para a outra saída desejada da Célula. Essa solução é viável tendo em vista que a quantidade de ciclos necessária para um dado atravessar a célula sendo operado é 3, enquanto a quantidade de ciclos para um dado roteado é de 2. Um estudo de atrasos e dependências de dados indica qual saída será menos prejudicada com esse atraso adicional de um ciclo.

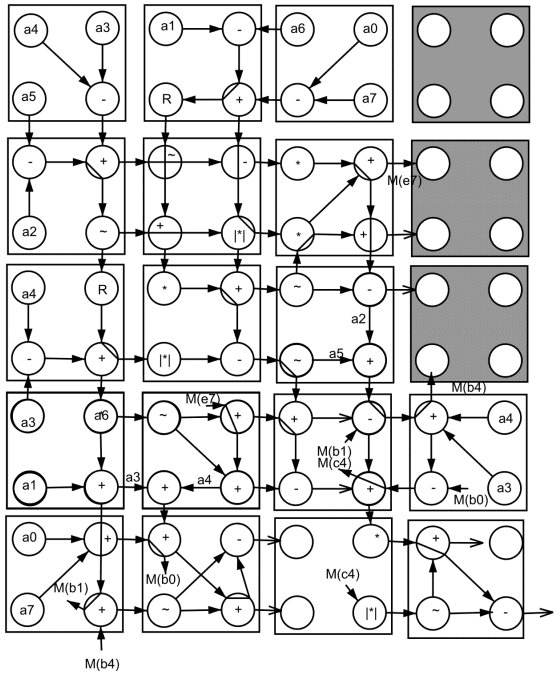

Figura 6. Implementação da DCT 1D

Devido a Unidade de Ponto Flutuante da ALU do processador ainda estar em fase de descrição em VHDL, e por causa da necessidade de se multiplicar um inteiro por constantes racionais, optou-se por multiplicar as constantes M_1 à M_4 por 256, de modo a trabalhar apenas com números inteiros. Os resultados das multiplicações das constantes foram arredondados e o resultado desse arredondamento foi utilizado em seu lugar. Dividindo-se esses novos valores por 256 tem-se as aproximações utilizadas, valores descritos na **Tabela 1** bem como os erros gerados por essa solução.

Com essas aproximações obteve-se um erro médio de 0,000962625. Este erro é aceitável tendo em vista que os resultados serão posteriormente usados na quantização e possivelmente sofreram um *downsampling*, processos que diminuem a precisão dos valores de saída. Os valores utilizados para as constantes podem ter uma maior precisão, o que implica em uma menor taxa de erro. Este fato e suas conseqüências na implementação serão abordados mais adiante nesta sessão.

O procedimento da utilização da multiplicação inteira criou um problema para o cálculo de F_2, F_3, F_4 e F_5, que somam/subtraem resultados que foram alterados pela multiplicação das constantes por 256 com resultados não modificados. Para contornar esse problema duas novas multiplicações foram inseridas ao fluxo de dados de modo a normalizar B7 e C4, valores gerados a partir apenas de somas/subtrações das entradas, fazendo com que esses valores ao serem somados ao resultado das multiplicações pelas constantes acima citadas conservem seus valores relativos. Na **Figura 6** essa normalização é representada pelo símbolo "|*|".

Tabela 1. Aproximações das contantes

M_1	
Valor Real:	0,707107
Valor Utilizado:	0,70703125
Erro:	0,00007575
M_2	
Valor Real:	0,382683
Valor Utilizado:	0,3828125
Erro:	0,0001265
M_3	
Valor Real:	0,541196
Valor Utilizado:	0,54296875
Erro:	0,00177275
M_4	
Valor Real:	1,306563
Valor Utilizado:	1,3046875
Erro:	0,0018755

Analisando-se o algoritmo apresentado nota-se que os valores que resultam em S_0 e S_4 não passam, no fluxo de dados, por nenhuma operação de multiplicação, ou seja, já estão em sua forma normal sendo prontos para uso imediato. Todavia isso não ocorre nos demais resultados, sendo necessária a sua divisão por 256 para que possa voltar a ter sua magnitude original. Como a unidade de divisão da ALU ainda não foi completamente implementada, foi necessária a utilização de um simples artifício para contornar este problema. Todos os resultados que ainda não estão na sua magnitude original são deslocados em 8 bits para a direita, o que equivale, em binário, a uma divisão por 256.

Sendo a largura das saída da DCT 2D de no máximo 15 bits, devido ao próprio padrão JPEG, pode-se utilizar um fator de multiplicação de 65536, que tem 16 bits de largura, aumentando consideravelmente a precisão dos resultados obtidos e fazendo uso da maior largura da palavra da arquitetura.

Como cada grão configurado como processador possui três Células que trabalham como ULA's dedicou-se 2 grãos (6 ULA's) para fazerem a volta à magnitude original dos seis resultados que trabalham em paralelo gerando um dado a cada 10 ciclos, 8 de deslocamentos, 1 de entrada na célula e outro de saída. Como dois resultados não precisam ser alterados tem-se 8 saídas a cada 10 ciclos, considerando-se o pipeline cheio. Dessa forma obtém-se uma matriz de 8x8 resultante da primeira DCT 1D a cada 80 ciclos.

3.3. A Implementação do Buffer de Transposição

O buffer de transposição foi implementado diretamente na programação dos grãos *Processadores*, de modo que a transposição seja feita na escrita da matriz nas posições da C-MEM. Alocam-se duas matrizes de 64 posições na C-MEM, uma para que nela sejam escritos os resultados da primeira DCT e a outra para servir de entrada para a segunda ou para, como se vê adiante, re-introduzir esses dados no pipeline. As funções dessas duas matrizes ficam se alternando. Quando a matriz para a escrita dos valores está cheia, esta passa a ser a matriz de leitura e a outra passa a ser a matriz de escrita.

3.4. Pipeline

As implementações de aplicações sistólicas em arquiteturas reconfiguráveis costumam fazer uso intenso de *pipelining*, e está não foge à regra.

Cada um dos resultados da DCT unidimensional configura um pipeline. Tem-se, assim, oito pipelines alocados na implementação.

Esses pipelines , por uma questão de economia de recursos, entrada/saída, e de espaço, não são totalmente independentes entre si.

A **Tabela 2** apresenta as latências e a quantidade de estágios em cada um dos pipelines.

3.5. Sincronização

Como os pares de saída têm latências diferentes faz-se necessário um mecanismo que garanta que sua ordem e freqüência não gerem incoerências nos dados, tanto nas matrizes transpostas no meio do fluxo quanto no resultado final.

Como já mencionado neste documento, a sincronização é feita por dependência de dados. Como é possível extrair um diagrama de atrasos da aplicação, é feito o cálculo da quantidade de ciclos necessários para cada resultado estar pronto e em que momento cada resultado estará disponível para o próximo estágio do pipeline. Com esses dados determina-se a melhor ordem e espaço entre as inserções dos dados no pipeline.

3.6. Implementações da Transformada Bidimensional

A implementação da DCT 2D em dois estágios de DCT's unidimensional nesta arquitetura reconfigurável, oferece duas possibilidades de implementação: uma voltada para um melhor desempenho e outra para uma menor área.

Tabela 2. Latências

Pipeline	Latência em ciclos	Estágios
S0	18	5
S4	17	5
S2	26	7
S6	29	8
S7	26	9
S1	24	7
S3	30	8
S5	33	9

3.6.1. Implementação Voltada para Desempenho

Na implementação voltada para desempenho utiliza-se o esquema da **Figura 4**, onde tem-se duas DCT's 1D interligadas por um buffer de transposição. O buffer de transposição, como explicitado anteriormente, foi encapsulado na programação dos grãos no Modo de Execução Processador, bastando replicar a arquitetura da DCT 1D. As saídas dos *Grãos Processadores* da primeira DCT 1D são direcionadas para o envio dos valores computados, já dispostos na matriz transposta, para suas respectivas entradas na segunda DCT 1D.

Os *Grãos Processadores* da segunda DCT têm um programa diferente, tendo em vista que nenhuma transposição é necessária, devendo ser garantida, com o programa a ser executado, a ordem de saída para o controle principal.

Essa implementação consome um total de 40 grãos gerando uma matriz de 8x8 a cada 80 ciclos, com o pipeline cheio.

3.6.2. Implementação Voltada para Economia de Área

Na implementação voltada para a economia de área utiliza-se apenas a implementação da DCT unidimensional apresentada na **Figura 6.** Para o cálculo da segunda DCT unidimensional os resultados obtidos pela primeira DCT, transpostos, são reintroduzidos no pipeline. Essa implementação mantem em 20 a quantidade de grãos necessários (**Figura 6**), mas diminui pela metade a freqüência de saída das matrizes de resultados, ou seja, gera uma matriz de 8x8 a cada 160 ciclos, com o pipeline cheio. Essa implementação também requer alteração do código dos *Grãos Processadores*, pois terão que tratar da alternância da fonte dos dados como também para onde são enviados, se serão reintroduzidos ou enviados para o controle principal.

4. RESULTADOS E COMPARAÇÕES COM OUTRAS IMPLEMENTAÇÕES

Nesta sessão serão apresentados os resultados conseguidos com as implementações acima apresentadas e comparações com implementações em FPGA e em

software da transformada discreta do cosseno em duas dimensões.

4.1. Resultados obtidos

A célula foi descrita em VHDL e sintetizada, com as limitações descritas no artigo, no Quartus II da Altera em FPGA da família Flex 10KE também da Altera obtendo os resultados da **Tabela 3**.

Tabela 3. Resultados da compilação

Dispositivo	EPF10K200EBC600-1
Total de LE´s	3217/9984 (32%)
Total de pinos	442/470 (94%)
Total de bits de EAB	34816/98304 (35%)
Clock	53,08 MHz

62Na implementação voltada a minimização da área o desempenho alcançado permite uma taxa de 70,7 imagens monocromáticas de 640x480 por segundo ou 27,65 em 1024x768. Na implementação voltada para desempenho tem-se o dobro do anterior, ou seja, 141,4 imagens de 640x480 e 55,3 de 1024x768 por segundo.

Em ambos os casos a latência inicial é de 160 ciclos.

4.2. Comparação com uma implementação ASIC

Em [9, 10] é descrita uma implementação em FPGA da mesma DCT 2D, também sintetizada na família FLEX10KE da Altera. O algoritmo utilizado nessa implementação é o mesmo aqui apresentado, bem como a estratégia de dividir o problema em duas DCT's unidimensionais com um buffer de transposição ligando-as.

Os erros nas arquiteturas comparadas, devido a aproximação das constantes M_1 à M_4, se equivalem por estarem por volta da terceira casa decimal, sendo que as implementações aqui propostas possui um erro médio 60% menor. A equivalência dos erros não é de forma alguma coincidência, foi a partir da análise dos erros que se chegou ao valor de 256 no multiplicando, pois assim tería-se uma qualidade equivalente nas saídas das duas arquiteturas, facilitando assim sua comparação.

Na arquitetura comparada a latência inicial é praticamente igual a aqui apresentada, 163 e 160, respectivamente. No que tange a saída de dados o X4CP32 tem uma pequena desvantagem, pois requer uma quantidade maior de ciclos para o cálculo da transformada, 80 contra 64 ciclos por matriz de 8x8 pixels da implementação ASIC. Levando-se em consideração a freqüência de operação das duas arquiteturas, observa-se que o X4CP32 obtém um ganho de desempenho final em torno de 14%, utilizando a implementação voltada para otimizar área. Comparada com a implementação voltada a desempenho, o ganho de desempenho passa a ser de 127%.

4.3. Comparação com Implementação em Software

O algoritmo apresentada foi também implementado em software e executado em um PC equipado com um processador Duron^{tm} 950 Mhz da AMD, 88 Mbytes de RAM e rodando o Linux Slackware 8.0. O algoritmo foi implementado em C++ e compilado com o g++ da GNU.

No software foi introduzido um loop de 1.000.000 de passos onde se calcula a DCT 2D. O programa foi executado e o tempo de CPU computado com a ajuda da ferramenta *time*.

Para o cálculo das 1.000.000 matrizes de 8x8 elementos foram consumidos 7,12s. Essa mesma quantidade de matrizes consumiria, na implementação voltada a desempenho, 1,5s, o equivalente a 21% do tempo da implementação em software.

5. CONCLUSÕES

Este artigo apresentou a implementação da Transformada Discreta do Cosseno em duas dimensões (DCT 2D), uma aplicação famosa pelo seu uso nos padrões MPEG e JPEG, na arquitetura reconfigurável X4CP32 em duas versões, assim como os resultados obtidos. Foram apresentadas comparações com um ASIC e uma implementação em software. Na versão voltada para a economia de área, o X4CP32 obteve um ganho final de 14% sobre o ASIC e de 137% sobre a implementação em software. Na versão voltada para o melhor desempenho, o ganho do X4CP32 sobre o ASIC chegou a 127%, e 375% sobre a implementação em software.

6. REFERÊNCIAS

[1] R. Hartenstein (invited keynote): "Reconfigurable Computing: a New Business Model - and its Impact on SoC Design"; *DSD'2001*, Warzaw, Poland, Sept 4 - 6, 2001

[2] R. Hartenstein (embedded tutorial): "A Decade of Research on Reconfigurable Architectures - a Visionary Retrospective"; *DATE 2001*, Munich, March 2001

[3] E. Waingold, M. Taylor, D. Srikishna, V.Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, A. Agarwal, "Baring It All to Software:Raw Machines", September -1997.

[4] E. Mirsky, A. DeHon, "Matrix: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources", *IEEE*, 1996

[5] B. Radunovic, “An Overview of Advances in Reconfigurable Computing Systems”, *32nd Hawaii International Conference on System Sciences,* 1999.

[6] R. Soares, A. Pereira, I. Saraiva, “X4CP32: a Programmable Multi-level Reconfigurable Microprocessor”, *Proceedings of Students Forum on Microeletronics 02*, SBC, Porto Alegre, pp. 69-73, 2002

[7] Y. Arai, T. Agui, M. Nakajima. “A fast DCT-SQ scheme form images” *Transactions of IECE*, vol. E71, n° 11, pp. 1095-1097, 1998

[8] M. Kovak,, N. Ranganathan, “Jaguar - A Fully Pipelined VLSI Architecture for JPEG Image Compression Standard. *Proceedings of IEEE*, vol. 83, n°.2, pp. 247-258, 1995

[9] Agostini, Luciano Volcan. “Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG”, *Porto Alegre: PPGC of UFRGS*, 2002.

[10] L. Agostini, I. Silva, S. Bampi. “Pipelined Fast 2D DCT Architecture for JPEG Image Compression”. *Proceedings of SBCCI,* pp 226-231, 2001.