Universidade Federal da Paraíba
Centro de Informática
Departamento de Informática
Programa de Pós-Graduação em Informática

Sistema Misto Reconfigurável Aplicado à Interface PCI para Otimização do Algoritmo *Non-local Means*.

Daniel Soares e Marques

Dissertação de Mestrado
João Pessoa, Paraíba – Agosto de 2012

Daniel Soares e Marques

Sistema Misto Reconfigurável Aplicado à Interface PCI para Otimização do Algoritmo *Non-local Means.*

Dissertação final apresentada ao Programa de Pós-Graduação em Informática do Departamento de Informática da Universidade Federal da Paraíba como requisito parcial para obtenção do título de mestre em informática.

Área de concentração: Sinais, Sistemas Digitais e Gráficos

Orientador: José Antônio Gomes de Lima

João Pessoa, Paraíba – Agosto de 2012

M357s Marques, Daniel Soares e.
Sistema misto reconfigurável aplicado à interface PCI para otimização do algoritmo *Non-Local Means* / Daniel Soares e Marques.-- João Pessoa, 2012.
100f. : il.
Orientador: José Antônio Gomes de Lima
Dissertação (Mestrado) – UFPB/CCEN
1.Informática. 2.*Non-local Means (NLM)*. 3. Processamento Digital de Imagens. 4. Computação reconfigurável. 5. PCI. 6. FPGA.

UFPB/BC CDU: 004(043)

AGRADECIMENTOS

Há muita gente envolvida no esforço conclusivo deste trabalho que gostaria de agradecer. Seria impossível lembrar o nome de todas as pessoas que seguiram minhas idéias, auxiliaram ou até se compadeceram em horas difíceis do rumo da pesquisa desenvolvida, servindo como verdadeiras válvulas de escape para a tensão e estresse acumulados. Contudo, há algumas pessoas que gostaria de agradecer especialmente.

Primeiramente, agradeço a Deus, a inteligência cósmica universal, que me tornou capaz de iniciar, desenvolver e concluir meu raciocínio durante os anos que me dediquei ao trabalho. O agradeço também por me ensinar dia após dia através da busca das coisas que realmente me são necessárias, verdadeiras e amáveis.

Também gostaria de agradecer à minha família. Agradeço à minha mãe, Maria Lúcia, que me ensina através do amor incondicional de uma mãe pelo filho. Agradeço a meu pai, Milton Marques, que me ensina através de sua disciplina e dedicação às atividades que ama. Agradeço à minha irmã, Anna Carolina, que me ensina através do maior exemplo de amor fraternal que conheço. Agradeço à minha filha, Clara Liz, que me ensina o amor de pai para filho.

Seguindo essa lógica, gostaria de agradecer muita gente que me auxiliou diretamente ou indiretamente, como amigos e colegas, cujo os nomes prefiro guardar comigo, com carinho no coração, pois a mente é falha e certamente iria esquecer de um ou outro importantíssimo para que o fruto desse mestrado germinasse.

Agradeço também à entidade financiadora deste estudo, o CNPq, juntamente ao corpo executivo da pós-graduação em informática da UFPB, em especial, meu orientador, José Antônio, por ter me aceitado como orientando e pelas força e motivação que me proporcionou, principalmente na etapa final incansável.

Por fim, agradeço ao tempo – o que tudo transforma – que me ensina a calma, a paciência... a ciência da paz.

Dedico este trabalho à minha família,
meus maiores professores.

“Devagar é que se anda. Devagar se chega lá.
Se cair, tu se levanta, continua a caminhar. [...]
Um passo de cada vez pra você não tropeçar...”

(Contramestre Barata – Capoeira Comunidade)

RESUMO

A área de processamento de imagens digitais está evoluindo continuamente e, embora as áreas de aplicações sejam diversas, os problemas encontrados comumente convergem para os métodos capazes de melhorar a informação visual para a análise e interpretação. Uma das principais limitações em questão de precisão de imagens é o ruído, que é definido como uma perturbação na imagem.

O método *Non-Local Means (NLM)* destaca-se como o estado da arte de filtragem de ruído. Contudo, sua complexidade computacional é um empecilho para torná-lo prático em aplicações computacionais de uso geral.

O presente trabalho apresenta a implementação de um sistema computacional, desenvolvido com partes executadas em software e em hardware aplicado à *PCI*, visando a otimização do algoritmo *NLM* através de técnicas de aceleração em hardware, permitindo uma eficiência maior do que normalmente é fornecida por processadores de uso geral.

O uso da computação reconfigurável auxiliou no desenvolvimento do sistema em hardware, proporcionando a modificação do circuito descrito no ambiente de sua utilização, acelerando a implementação do projeto. Utilizando um kit *PCI* de prototipação *FPGA*, para efetuar o cálculo dedicado da *Distância Euclidiana Quadrática Ponderada*, os resultados obtidos nos testes exibem um ganho de tempo até 3.5 vezes maior que as abordagens de otimização comparadas, mantendo também a qualidade visual da filtragem estabilizada.

Palavras Chave: Non-local Means; NLM; Processamento Digital de Imagens; Computação Reconfigurável; PCI; FPGA; ALTERA;

ABSTRACT

The digital image processing field is continually evolving and, although the diverse application areas, the commonly problems found converge to methods capable to improve visual information for analysis and interpretation. A major limitation issue on image precision is noise, which is defined as a perturbation in the image.

The *Non-Local Means (NLM)* method stands out as the state-of-the-art of digital image denoising filtering. However, its computational complexity is an obstacle to make it practical on general purpose computing applications.

This work presents a computer system implementation, developed with parts implemented in software and hardware applied to *PCI*, to optimize the *NLM* algorithm using hardware acceleration techniques, allowing a greater efficiency than is normally provided by general use processors.

The use of reconfigurable computing helped in developing the hardware system, providing the modification of the described circuit in its use environment, accelerating the project implementation. Using an *FPGA* prototyping kit for PCI, dedicated to perform the dedicated calculation of the *Squared Weighted Euclidean Distance*, the results obtained show a gain of up to 3.5 times greater than the compared optimization approaches, also maintaining the visual quality of the denoising filtering.

Key Words: Non-local Means; NLM; Digital Image Processing; Reconfigurable Computing; PCI; FPGA; ALTERA;

SUMÁRIO

LISTA DE FIGURAS

LISTA DE EQUAÇÕES

LISTA DE TABELAS

I. Introdução

1.1. Motivação

O método de filtragem de imagens *Non-local Means*, mais conhecido como *NLM*, foi estabelecido como o método mais eficiente para a remoção de ruídos em imagens binárias (SHAHAM, 2007), porém sua alta complexidade temporal o torna impraticável para aplicações computacionais usuais.

Observando a alta complexidade temporal do *NLM*, mesmo sob sua forma já otimizada em software comentada em (SAPIRO e MAHMOUDI, 2005), houve a idéia de uma investigação que conduzisse a uma otimização mais eficiente do algoritmo *NLM*, unindo partes processadas em hardware e software. Tal implementação visa contribuir com a aceleração do algoritmo original para tornar seu uso possível em aplicações comuns de processamento de imagens e vídeos digitais.

Com os conhecimentos adquiridos através da experiência de projetos utilizando princípios de computação reconfigurável, partindo também das afirmações feitas por (SOUZA, 2008), (ATHANAS e SILVERMAN, 1993) e (OLUKOTUN, HELAIHEL, *et al.*, 1994) relacionadas às vantagens da utilização de computação reconfigurável para projetos de alta performance computacional; agregando as abordagens sugeridas por (BONATO, MAZZOTI e MARQUES, 2009 apud GAMBARRA, 2010) para o desenvolvimento de projetos de sistemas com partes em software e hardware; observando o comentário em (GONZALES e WOODS, 2000) sobre a necessidade de hardware especializado para incrementar a velocidade de aplicações em processamento digital de imagens; unindo a complexidade de tempo de execução do algoritmo *NLM*, observando as vantagens descritas em (BUADES, COLL e MOREL, 2004) com relação ao uso de janelas de busca para efetuar o processamento da filtragem; surgiu, então, a idéia de desenvolvimento de um sistema misto reconfigurável aplicado à interface *PCI* para otimização do algoritmo *Non-local Means*.

1.2. Objetivos Gerais

O objetivo principal do trabalho é o desenvolvimento de um sistema misto reconfigurável aplicado à interface *PCI* para otimização do algoritmo *Non-local Means*.

Para a conclusão do trabalho várias atividades intermediárias foram cumpridas, dentre os quais podemos ressaltar principalmente:

- Estudo do algoritmo de filtragem de imagens digitais *NLM*;
- Desenvolvimento da parte de software do algoritmo *NLM;*
- Estudo do kit de prototipação *FPGA* ideal para o trabalho;
- Estudo do padrão de *handshake AMBA AXI*;
- Estudo de métodos de paralelismo em projetos de hardware;
- Descrição em hardware da parte do circuito do algoritmo *NLM;*
- Estudo e desenvolvimento do *driver* de comunicação entre as partes de software e hardware do sistema;
- Testes sobre o protótipo desenvolvido para apuração dos resultados.

1.3. Organização

No capítulo II será apresentada a fundamentação teórica necessária para o desenvolvimento do trabalho, incorporando conhecimentos sobre:

- Processamento digital de imagens e o *NLM;*
- Computação reconfigurável;
- Sistemas embarcados;
- Técnicas de paralelismo em hardware;
- Protocolo *AMBA AXI* para *handshake*.

O capítulo III exibe a idéia da proposta do sistema, foco deste trabalho. O capítulo IV exibe as ferramentas e métodos empregados para se atingir os resultados apresentados, e comparados com os resultados de outros trabalhos, no capítulo V. Por fim, o capítulo VI conclui o texto da dissertação fazendo considerações sobre o trabalho.

II. Fundamentação Teórica

O presente capítulo define os conceitos essenciais envolvidos no desenvolvimento do trabalho proposto na dissertação.

Inicialmente, é importante introduzir os conceitos referentes a processamento digital de imagens, mais especificamente o problema da eliminação de ruídos e o estudo do algoritmo *Non-local Means* – o *NLM* – , tema central do estudo elaborado neste trabalho. Neste ponto também são citadas bibliografias relacionadas ao assunto da otimização do algoritmo *NLM*.

Outro tema relacionado com o foco do trabalho é o projeto de sistemas que possuem partes desenvolvidas em software e em hardware. Esta abordagem típica de projetos de sistemas embarcados é discutida logo em seguida.

Por fim, é importante ter o conhecimento de elementos úteis para inserir no projeto de hardware com o intuito de facilitar o reuso dos blocos utilizados e atingir o melhor desempenho possível. Neste sentido, se observou a necessidade do estudo de abordagens de paralelismo de estruturas de hardware e processo de *handshake* entre blocos lógicos. Dessa forma, esse tema aborda os conteúdos teóricos fundamentais para a compreensão daquilo utilizado para a realização da abordagem de otimização do algoritmo *NLM* pretendida.

2.1. Processamento Digital de Imagens

Conforme os sistemas digitais foram evoluindo ao longo dos anos, a utilização dos mesmos foi sendo expandida para áreas diversas. O processamento de sinais, antes efetuado apenas através de métodos analógicos, também é uma das áreas do conhecimento que passou a ser resolvido por meio de processamento digital.

Focando-se em processamento de sinais digitais, ao tratar-se de sinais pictóricos, ou seja, imagens digitais, se está pondo em evidência um sinal que pode ser representado por uma função bidimensional capaz de ser processada por um computador, por exemplo.

Essa área do conhecimento é aplicada a qualquer computação na qual a entrada é uma imagem, que será processada por procedimentos geralmente expressos de forma algorítmica, e cujo resultado de saída será uma nova imagem que é a imagem processada. Dessa forma, segundo (GONZALES e WOODS, 2000), a maioria das funções de processamento de imagens pode ser implementada em software. A única razão para hardware especializado para processamento de imagens é a necessidade de velocidade em algumas aplicações ou para vencer algumas limitações fundamentais da computação.

A área de processamento de imagens digitais está evoluindo continuamente e embora as áreas de aplicações sejam diversas, tais problemas comumente convergem para os métodos capazes de melhorar a informação visual para a análise e interpretação (GONZALES e WOODS, 2000).

2.1.1. Imagens Digitais e Ruído

Uma imagem digital é geralmente codificada como uma matriz de níveis de cinza ou valores de cor composta em tons de vermelho, verde e azul, que satisfazem o padrão *RGB* (*Red, Green and Blue*).

Uma das principais limitações em questão de precisão de imagens é o ruído, que é definido como uma perturbação na imagem. Por exemplo, para ruído com componente aditivo, tem-se a Equação 2.1. Já a Equação 2.2 exibe o exemplo para ruído com componente multiplicativo:

$$v(i) = u(i) + n(i)$$

Equação 2.1. Valor observado num pixel *i* dado um ruído com componente aditivo.

$$v(i) = u(i)\ n(i)$$

Equação 2.2. Valor observado num pixel *i* dado um ruído com componente multiplicativo.

onde:

- *v(i)*: Valor observado;
- *u(i)*: Valor real no pixel *i*;
- *n(i)*: Valor de perturbação (ruído) no pixel *i*.

Nestes exemplos, a redução da componente de ruído na composição implica no aumento da qualidade da imagem.

Grande parte dos métodos para se eliminar ruído de imagens, descritos de forma algorítmica, depende de um parâmetro de filtragem *h* que é determinado como uma função do desvio padrão do ruído. Desse modo, a imagem final poderá ser descrita como a combinação de dois componentes, a imagem suavizada e a componente de ruído (SHAHAM, 2007) conforme a Equação 2.3:

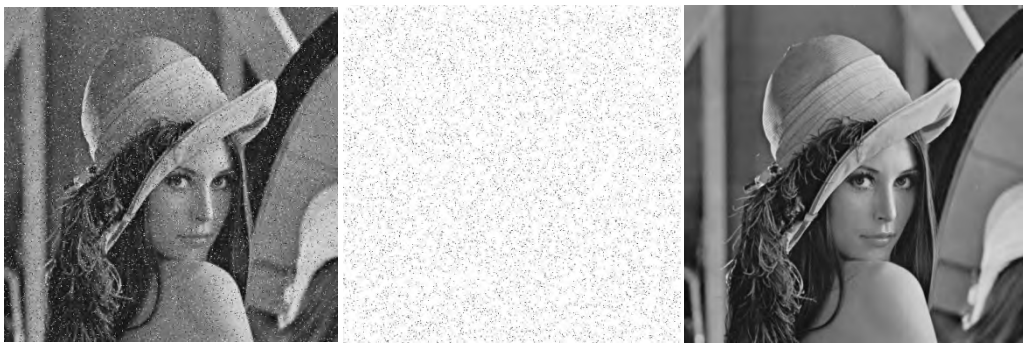
$$v = D_h v + n(D_h, v)$$

Equação 2.3. Imagem final composta de uma parte suavizada e uma parte com ruído com componente aditivo.

onde:

- *v*: Imagem original, inicialmente com ruídos;
- D_h: Método de remoção do ruído;
- $n(D_h,v)$: Parte da imagem original que o método reconheceu como ruído;
- $D_h v$: Imagem *v* suavizada através do método D_h.

O ideal é que a componente $D_h v$ seja mais suave que *v*. A Figura 2.1 mostra o exemplo de uma imagem (*v*), seu componente de ruído (*n*) e seu componente filtrado ($D_h v$).



(a) **(b)** **(c)**

Figura 2.1. (a) Imagem *v* inicialmente com ruído, (b) Componente de ruído da imagem *v*, (c) Componente sem ruído da imagem *v*.

Um dos grandes problemas envolvendo algoritmos de remoção de ruídos em imagens é que os mesmos não diferem pequenos detalhes da imagem do ruído a ser tratado e, portanto, acabam por remover partes da imagem que não são ruídos. Todos os métodos assumem que o ruído na imagem é oscilatório e que a imagem é suave, ou parcialmente suave. Dessa forma, tentam separar o que é suave do que é oscilatório, porém muitas estruturas inerentes à imagem são tão oscilatórias quanto um ruído. Com isso, em muitos casos os algoritmos geram novas distorções na imagem (BUADES, COLL e MOREL, 2004).

Conforme (WANG e BOVIK, 2006) talvez a mais simples e mais utilizada medida objetiva para qualidade de imagens seja o *Erro Médio Quadrático* (*MSE – Mean Squared Error*).

Sejam $x = \{x_i \mid i = 1, 2, ..., N\}$ e $y = \{y_i \mid i = 1, 2, ..., N\}$ representações de duas imagens a serem comparadas, onde N é o número de pixels e x_i e y_i são intensidades do i-ésimo pixel nas imagens x e y, respectivamente, assumindo que x seja a imagem original, sem ruídos, e y a imagem distorcida, cuja qualidade esteja sendo avaliada. Então o *MSE* e o índice *PSNR (Peak Signal-to-Noise Ratio)* – outra medida de qualidade relacionada ao *MSE* e geralmente utilizada – são respectivamente definidos nas Equações 2.4 e 2.5:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

Equação 2.4. Cálculo do *MSE* entre as imagens *x* e *y*.

$$PSNR = 10 \log_{10} \frac{L^2}{MSE}$$

Equação 2.5. Cálculo do índice *PSNR* entre as imagens *x* e *y*.

onde:

- *L:* Alcance máximo dos níveis de cores da imagem. Para imagens com 8 bits de resolução de níveis de cinza, esse valor será 255.

2.1.2. O Algoritmo *Non-local Means*

O algoritmo *Non-local Means* (*NLM*) foi proposto por (BUADES, COLL e MOREL, 2004) e tem como objetivo a filtragem de ruídos em imagens digitais.

Assim como a maioria dos algoritmos para redução de ruídos, o algoritmo *NLM* utiliza cálculo de médias como método para remover o ruído em imagens digitais. Contudo, enquanto a abordagem utilizada pela maioria dos algoritmos convencionais segue a idéia de que as características que estão próximas entre si tendem a possuir valores semelhantes e, portanto, são usadas para calcular a média, o *NLM* realiza uma filtragem não local, onde, para o valor de cada pixel, os valores de todos os pixels da imagem são levados em consideração, assumindo que imagens naturais possuem características que se repetem globalmente e não apenas localmente. Dessa forma, o algoritmo utiliza um grau de relevância entre todos os pixels, baseando-se na semelhança entre os mesmos, para efetuar a filtragem proposta, mesmo que estes pixels não estejam na vizinhança daquele pixel a ser filtrado. Com isso, a filtragem de um pixel *p* em questão é efetuada através de uma média ponderada de todos os pontos da imagem. A Figura 2.2 abaixo demonstra um exemplo para ilustrar a idéia de relevância entre pixels.

Figura 2.2. Janelas de relevância entre pixels.

Na imagem da Figura 2.2 há quatro regiões separadas, denominadas *P1*, *P2*, *P3* e *P4*. De acordo com o cálculo do *NLM*, apesar da região *P2* estar mais próxima da região *P1*, as regiões *P3* e *P4* mostram-se mais relevantes para a filtragem de *P1*. Isso ocorre devido ao grau de semelhança entre as regiões citadas. Efetuando-se um zoom envolvendo aquelas regiões podemos observar mais precisamente a semelhança entre tais janelas, como é mostrado na Figura 2.3 abaixo.

Figura 2.3. Zoom sobre as janelas de pixels.

Dessa forma, o algoritmo *NLM* mostra que apesar de existir uma grande probabilidade de se achar pixels semelhantes em uma vizinhança próxima, também existe a possibilidade de se encontrar pixels relevantes localizados a uma maior distância.

2.1.2.1. Descrição do Algoritmo

Inicialmente o algoritmo recebe uma imagem de entrada e para cada pixel dessa imagem é efetuado um cálculo para determinar a influência – denominada “peso” – de cada um dos outros pixels em relação a este pixel que está sendo verificado. Segundo (BUADES, COLL e MOREL, 2004), os pesos são atribuídos utilizando o cálculo da

Distância Euclidiana Quadrática Ponderada. Em (EFROS e LEUNG, 1999) prova-se que esta métrica é adequada e consistente em um ambiente com ruído.

Dada uma imagem com ruído $v = \{v(i) | i \in I\}$, onde I é domínio da imagem, o valor estimado da filtragem efetuada pelo *NLM* em um pixel nessa imagem é computado como sendo a média ponderada de todos os pontos da imagem, como demonstrado pela Equação 2.6:

$$NL(v)(i) = \sum_{j \in I} w(i,j)v(j)$$

Equação 2.6. Cálculo do *NLM*.

onde:

- $NL(v)(i)$: Valor resultante da filtragem *NLM* do pixel i;
- $w(i,j)$: Peso entre os pixels i e j. Valor dentro do intervalo [0,1]. A soma de todos os pesos é 1;
- $v(j)$: Valor da intensidade de cor do pixel j na imagem original.

Um sistema de vizinhança para I é uma família $N = \{ Ni \}_{i \in I}$ de subconjuntos de I tal que, para todo $i \in I$ (BUADES, COLL e MOREL, 2004) :

- $i \in Ni$
- $j \in Ni \Longrightarrow i \in Nj$

O subconjunto Ni é dessa forma denominado de vizinhança para o pixel i. As vizinhanças podem ter tamanhos e formatos distintos, mas por simplicidade uma janela retangular é a abordagem mais utilizada. A semelhança entre os pixels i e j depende da semelhança entre os níveis de cinza, dada pelos vetores $v(Ni)$ e $v(Nj)$. Como exemplo, voltando para a Figura 2.3, as vizinhanças *P1* a *P4* são dadas pela coleção dos níveis de cinza de todos os pixels nos quadrados que os circundam. Assim, os pontos com vizinhanças de níveis de cinza similares a $v(Ni)$ terão pesos maiores no cálculo da filtragem sobre o ponto i. Conforme (BUADES, COLL e MOREL, 2004), a semelhança entre os vetores de níveis de cinza pode ser computada através do cálculo da *Distância Euclidiana Quadrática Ponderada*, doravante *Distância* L^2. O cálculo da *Distância* L^2 é definido na Equação 2.7:

$$d_a^2(x,y) = \sum_{m=1}^{n} a_m (x_m - y_m)^2$$

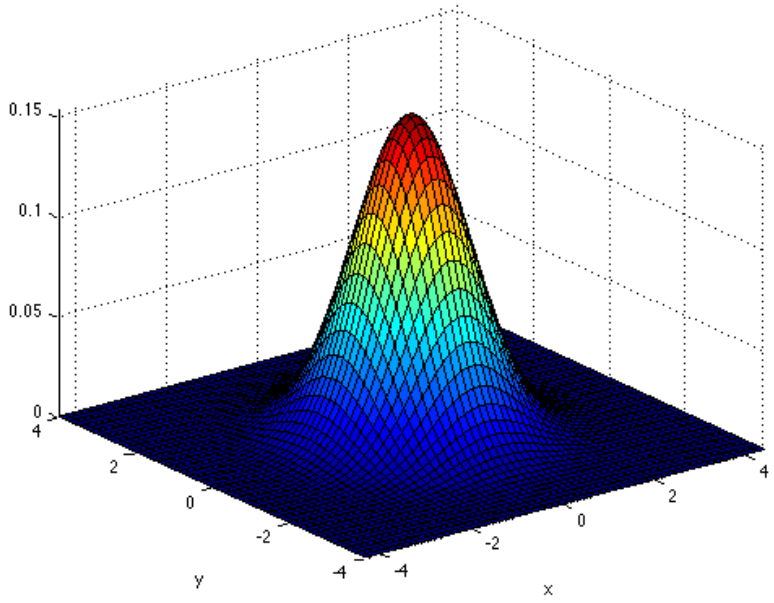
Equação 2.7. Cálculo da *Distância Euclidiana Ponderada Quadrática* (*Distância L^2*).

onde:

- $d^2_a(x,y)$: Valor do cálculo da *Distância L^2* entre os vetores x e y;
- a_m: Coeficiente de ponderação do termo m do somatório;
- x_m e y_m: m-ésimos índices dos vetores x e y.

Para o problema do *NLM*, o coeficiente a_m representa a importância de cada pixel na janela de vizinhança e é definido de acordo com um *Kernel Gaussiano* bidimensional.

O gráfico do *Kernel Gaussiano* bidimensional em sua forma isotrópica (*i.e.* circularmente simétrica) está exibido como parte da Figura 2.4, um *kernel* de tamanho 5x5 completa o conteúdo da imagem. Quando o *kernel* é aplicado à vizinhança centrada no ponto *p*, o ponto central (*p*) recebe o maior peso e os outros pontos da vizinhança recebem peso exponencialmente inverso a sua distância à *p*.



$$\frac{1}{273}$$

1	**4**	**7**	**4**	**1**
4	**16**	**26**	**16**	**4**
7	**26**	**41**	**26**	**7**
4	**16**	**26**	**16**	**4**
1	**4**	**7**	**4**	**1**

Figura 2.4. Representação da função gaussiana em duas dimensões com média em (0.0) e desvio padrão σ = 1.0.

De acordo com o exemplo da Figura 2.4 o cálculo será efetuado levando-se em consideração janelas com vizinhança 5x5, onde o peso de cada pixel da janela de vizinhança é determinado com a ajuda dos valores da matriz do *Kernel Gaussiano* bidimensional 5x5, que atribui maior influência para os pixels que estiverem mais próximos do pixel central, como pode-se perceber acima.

Assim, sendo *Ni* e *Nj* janelas de semelhança centradas nos pixels *i* e *j*, respectivamente, que se correspondem, é possível considerá-las vetores de intensidade de nível de cinza com a mesma cardinalidade do *Kernel Gaussiano*. Portanto, a equação da *Distância* L^2 poderá ser reescrita da seguinte forma:

$$\left\| v(Ni) - v(Nj) \right\|_{2,a}^2 = \sum_{m=1}^{n} a_m (v(Ni_m) - v(Nj_m))^2$$

Equação 2.8. *Distância L^2* reescrita para vetores de níveis de cinza.

Definido o cálculo da *Distância L^2*, o peso de cada pixel é determinado de acordo com a seguinte equação:

$$w(i,j) = \frac{1}{Z(i)} e^{(-1)\frac{\left\| v(Ni) - v(Nj) \right\|_{2,a}^2}{h^2}}$$

Equação 2.9. Cálculo do peso entre os pixels *i* e *j*.

Onde *Z(i)* é determinado como sendo o fator de normalização, definido por:

$$Z(i) = \sum_{j \in i} e^{(-1)\frac{\left\| v(Ni) - v(Nj) \right\|_{2,a}^2}{h^2}}$$

Equação 2.10. Cálculo do fator de normalização para o pixel *i*.

O termo *h* foi definido como fator de decaimento, responsável por controlar o decaimento dos pesos em função da distância euclidiana calculada. Quanto menor o valor de *h* menor será a contribuição dos pixels que apresentam uma distância euclidiana com valor alto, e quanto maior o valor de *h* maior será a influência desses pixels no cálculo final do algoritmo *NLM*. Caso tenha-se um valor de *h* muito alto, teremos uma maior distorção na imagem após o processamento da filtragem, e se o valor for muito baixo o ruído pode não ser removido suficientemente. Nos testes efetuados em (SHAHAM, 2007) foi decidido o uso de $h^2 = 200$ que após análise, como mostrado na Figura 2.5, apresentou um bom compromisso entre a remoção do ruído e distorção da imagem.

Figura 2.5. Efeito do fator de decaimento na imagem filtrada.

Na Figura 2.5, é possível observar o efeito do fator de decaimento. (*a*) mostra uma imagem com ruído de desvio padrão $\sigma_n = 20$. As três outras imagens são imagens geradas através da execução do algoritmo *NLM* com fatores de decaimento diferentes. A imagem (*b*) possui $h^2 = 200$; (*c*) possui $h^2 = 100$; e, por último, (*d*) possui $h^2 = 2000$.

2.1.2.2. Complexidade do *NLM*

Um grande problema que impede o uso do *NLM* em aplicações usuais em processamento de imagens é a sua alta complexidade computacional. Muito embora o *NLM* seja considerado um bom filtro para remoção de ruído, possuindo desempenho superior a outros algoritmos para eliminação de ruídos em imagens, principalmente quando se trata do ruído branco gaussiano como descrito em (BUADES, COLL e MOREL, 2004) e (GOOSSENS, LUONG, *et al.*, 2010), a complexidade computacional do algoritmo original é $O(n^2w)$, onde n é referente ao número de pixels presentes na imagem e w é o tamanho da vizinhança referente ao *Kernel Gaussiano* bidimensional.

De uma forma mais simples, essa complexidade se deve ao fato de que para calcular o valor filtrado de cada ponto da imagem deve-se varrer todos os o pontos (*n*) e calcular a *Distância* L^2 (*w*). Tal complexidade é tão alta que torna seu uso impraticável num contexto utilizando uma arquitetura computacional de um *Pentium IV* com frequencia de operação de 3 *GHz* em simples imagens de dimensão de 512 x 512, levando aproximadamente 5 horas de execução (DAUWE ET AL., 2008).

Para diminuir a complexidade do algoritmo, o *NLM* define a utilização de janelas de busca dentro da qual a busca por vizinhanças semelhantes será efetuada. Isto permite reduzir a complexidade de tempo para *O(nsw)*, onde *s* é o tamanho da janela de busca. Contudo, esta redução ainda possui complexidade computacional muito alta comparada a típicas abordagens de filtragem de imagens digitais, porém o uso desta abordagem melhora o desempenho visual em áreas uniformes, quando comparado com o algoritmo *NLM* original.

Utilizando-se uma janela de busca em lugar de pesquisar toda a imagem, o algoritmo abandona a idéia de filtrar utilizando características semelhantes na imagem que estão fora da janela de busca. A Figura 2.6 demonstra o uso dessa abordagem.

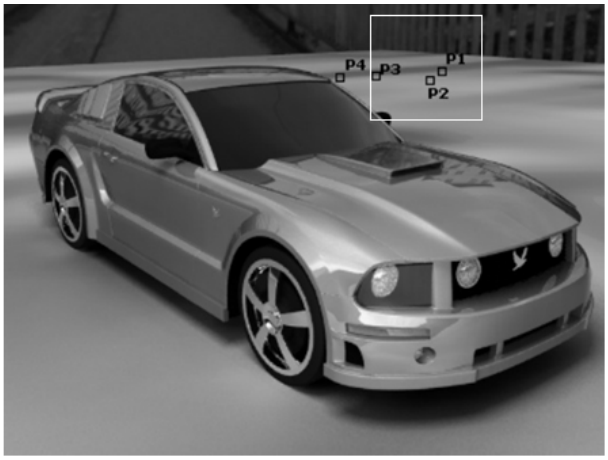


Figura 2.6. Uso de uma janela de busca para o cálculo do *NLM*.

Dessa forma, para o cálculo do *NLM* sobre um pixel centrado na janela de busca demonstrada acima, das janelas *P1 a P4* em evidência, apenas a janela *P4* estará fora do cálculo da filtragem efetuada pelo algoritmo. Porém, como dito anteriormente, mesmo modificando características do algoritmo original, o algoritmo modificado não reduz o tempo de processamento a níveis razoáveis.

2.1.2.3. O *NLM* e Outros Métodos de Filtragem de Ruídos

Apesar da sofisticação dos métodos anti-ruídos propostos mais recentemente, grande parte ainda não atingiu níveis desejáveis de aplicabilidade. Todos mostram boas performances quando o modelo da imagem a ser tratada obedece à aplicabilidade do algoritmo, mas no geral falham e criam artefatos do método ou removem estruturas inerentes à imagem e não o ruído. Pensando nestas questões, em (BUADES, COLL e MOREL, 2004) há a definição de uma metodologia matemática e experimental para comparar e classificar algoritmos clássicos de filtragem de imagens. Tal análise matemática baseia-se na análise do *ruído de método*, definido como a diferença entre a imagem digital e sua versão filtrada. Os métodos comparados são:

- Filtro Gaussiano (*GF*);
- Deslocamento de Curvatura Média;
- Variação Total (*TV*);
- Filtro Anisotrópico (*AF*);
- Variação Total Iterada (*TV Iter*);
- Filtragem de Vizinhança *Yaroslavsky* (*YNF*);
- *Translation Invariant Wavelet Thresholding* (*TIWT*);
- *DCT empirical Wiener filter* (*EWF*).

A performance de filtragem de cada método em questão foi comparada de acordo com:

- A magnitude do ruído de método;
- Artefatos do algoritmo e violação do modelo da imagem;
- Cálculo do erro médio quadrático entre as imagens filtradas e as originais;
- Avaliação visual dos resultados do ruído de método. Quanto mais o ruído de método for semelhante ao ruído branco, melhor será o método da filtragem. Essa forma de avaliação é considerada a melhor.

A Figura 2.7 mostra as imagens dos ruídos de métodos gerados por cada método de filtragem. Neste caso, quanto mais parecido com o ruído branco, melhor será o método utilizado na filtragem. Na imagem, tem-se: (*a*) *Filtragem Gaussiana*; (*b*) *Deslocamento de Curvatura Média;* (*c*) *Variação Total;* (*d*) *Variação Total Iterada;* (*e*) *Filtragem de Vizinhança;* (*f*) *Hard TIWT;* (*g*) *Soft TIWT;* (*h*) *DCT; e* (*i*) *NLM*. Neste caso, a imagem (*i)*, referente à filtragem *NLM* é a que se comporta com o melhor resultado.

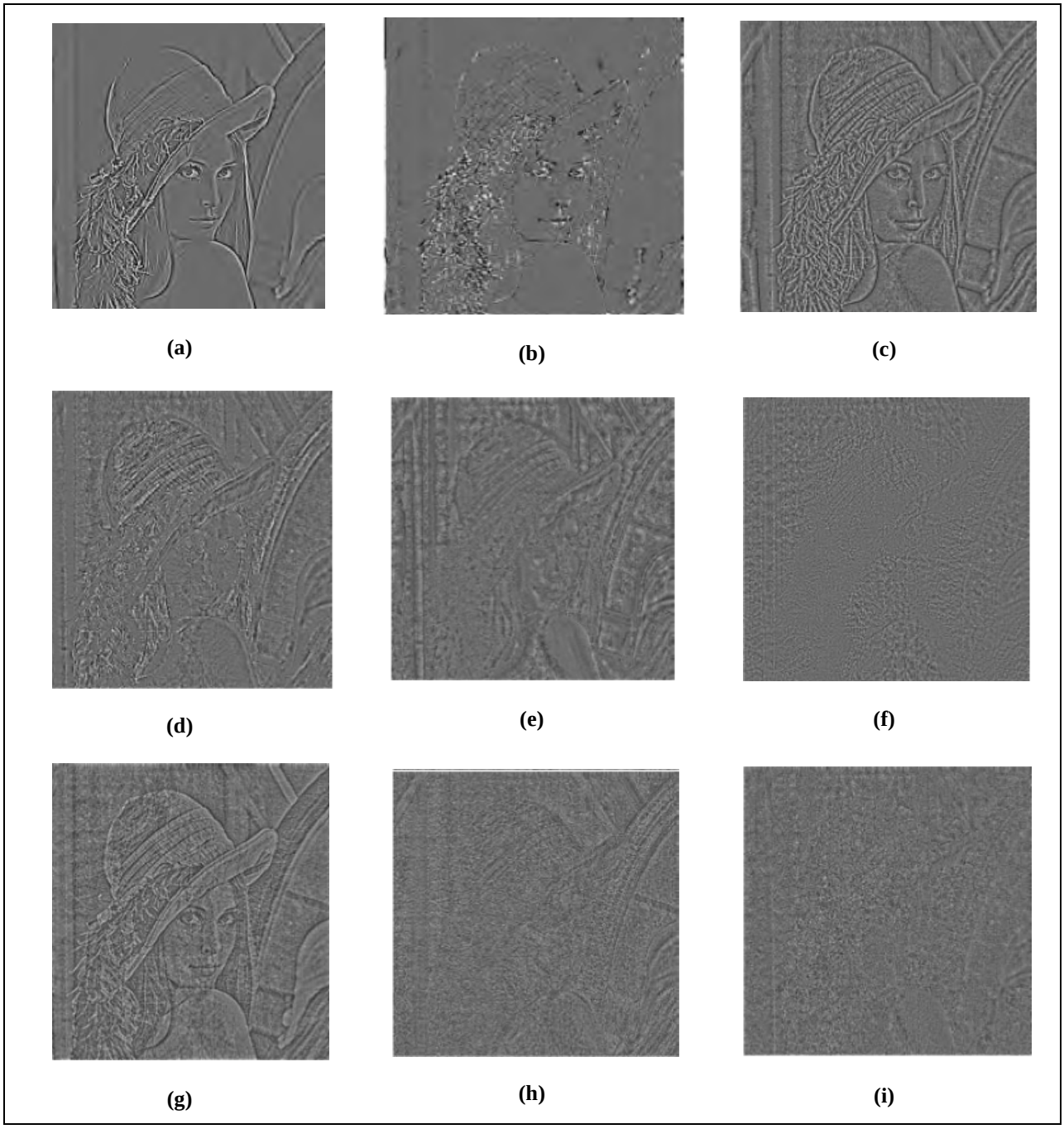


Figura 2.7. Ruídos dos métodos utilizados.

A Tabela 2.1 exibe um comparativo dos valores de *MSE* de cada método envolvido para um conjunto de imagens-teste com ruído embutido.

A primeira coluna refere-se à lista de imagens utilizadas, indicando o desvio padrão σ do ruído presente nela. As colunas seguintes referem-se aos métodos de filtragem utilizados citados anteriormente. Quanto menor o valor do erro médio

quadrático, mais próximo da imagem original a imagem filtrada resultante está. Os valores devem ser comparados por linha.

Tabela 2.1. MSE dos método de filtragem testados (BUADES, COLL e MOREL, 2004).

Imagem	σ	GF	AF	TV	TV Iter	YNF	EWF	TIHWT	*NLM*
Boat	8	53	38	39	29	39	33	28	23
Lena	20	120	114	110	82	129	105	81	68
Barbara	25	220	216	186	189	176	111	135	72
Baboon	35	507	418	385	364	381	396	365	292
Wall	35	580	660	721	715	598	325	712	59

Como é possível observar pelos resultados alcançados demonstrados na Tabela 2.1, tanto no cálculo do *MSE* quanto na imagem de ruído de método, o *NLM* foi o método que obteve melhores resultados. A única desvantagem do *NLM* é a sua complexidade de execução que acarreta no elevado tempo de processamento do método de filtragem.

2.1.3. Trabalhos Relacionados a Otimizações do *NLM*

Há na literatura científica atual uma vasta bibliografia referente a trabalhos que procuram otimizar o algoritmo *NLM*. Contudo, são visíveis as diferenças, tanto de otimização quanto de testes efetuados, entre as abordagens empregadas nos trabalhos disponíveis.

Em (SAPIRO e MAHMOUDI, 2005) há bons resultados de aceleração da execução do *NLM* original, sem o uso de janelas de busca. Neste trabalho o método é acelerado através da introdução de filtros que eliminam vizinhanças sem correlação durante o cálculo da *Distância* L^2 na janela de busca. Tais filtros são baseados em médias locais, dos valores dos pixels, e gradientes, gerando uma pré-classificação das vizinhanças a serem utilizadas que reduz a complexidade quadrática, do tempo de execução do *NLM*, para uma complexidade linear. Esta abordagem implica na redução da influência de áreas com grau de semelhança baixo durante a filtragem de um dado pixel.

Já (LAI e DOU, 2010), propõe otimizações no *Kernel Gaussiano* e uma abordagem de cálculo de pré-classificação da similaridade da vizinhança entre pixels, resultando na melhoria da filtragem e preservação dos detalhes da imagem. Apesar de mostrar valores consideráveis de qualidade de filtragem, por meio do índice *PSNR*, e afirmar que o tempo da computação é semelhante ao demonstrado em (SAPIRO e MAHMOUDI, 2005), não são citadas informações sobre as imagens de teste utilizadas.

Em (AVANAKI, DIYANAT & SODAGARI, 2008) o trabalho promete encontrar melhorias no cálculo do *NLM* com relação ao *fator de decaimento* (*h*) que conduzem a otimizações na performance de filtragem. Neste trabalho há vários resultados de testes sobre imagens, com índices de tempo de execução e qualidade de imagem expostos, porém também não cita informações sobre imagens de teste.

Já em (BILCU e VEHVILAINEN, 2008) é proposto uma otimização (*ICINLM)* baseada num método denominado *Interseção de Intervalos de Confiança* (*Intersection of the Confidence Intervals*), que procura manter o desempenho de filtragem do *NLM* convencional porém com baixo custo computacional através de cálculos baseados em aproximação polinomial local. Nos testes definidos neste trabalho são apresentadas melhorias consideráveis, de tempo de execução e de qualidade da filtragem, quando comparado com o proposto em (BUADES, COLL e MOREL, 2004) e (SAPIRO e MAHMOUDI, 2005).

A abordagem de otimização descrita em (PANG et al., 2009) – baseada em pré-seleção de pixels para o cálculo da similaridade entre os pixels – também apresenta bons resultados de otimização, inclusive quando comparado com o trabalho desenvolvido em (BILCU e VEHVILAINEN, 2008).

Para se ter uma idéia melhor do poder das otimizações envolvidas, o ideal seria implementar cada um dos trabalhos propostos para efetuar as comparações com maior controle sobre as variáveis de teste em questão, o que não é tão viável devido ao tempo gasto para se concluir tal atividade. Dessa forma, a comparação entre trabalhos correlatos deve ser efetuada com cautela, observando as diferentes técnicas empregadas e os resultados perceptíveis das imagens.

O trabalho dessa dissertação se baseia na abordagem descrita em (GAMBARRA, 2010), que utiliza a idéia de janelas de busca descrita em (BUADES, COLL e MOREL, 2004). Essa abordagem foi selecionada por ser preparada para utilização em hardware digital e por sua verificação funcional já estar validada.

2.2. Computação Reconfigurável

Nos últimos anos, o cenário da microeletrônica mundial tem evoluído bastante e atualmente é comum observar à volta uma variedade de dispositivos eletrônicos cada vez mais velozes, com capacidades de armazenamento de dados maiores, consumo de energia e custos de mercado mais baixos. O balanço entre velocidade de operação e generalidade desses sistemas é o desafio para os projetistas de computadores (VILLASENOR e SMITH, 1997).

Com a evolução das tecnologias de semicondutores é possível projetar sistemas mais gerais em chips, que executem grandes quantidades de cálculos diferentes. Antes de converter um projeto de circuito digital em um *CI* concreto, a indústria requer algumas semanas ou até meses para chegar ao projeto finalizado no silício. Entretanto, durante as fases intermediárias do projeto, é recomendada a utilização de alguma tecnologia de circuitos integrados reconfigurável (VAHID, 2008). Com isso, testes práticos podem ser efetuados sobre o projeto do dispositivo implementado antes do mesmo ser enviado para a indústria de circuitos integrados para a finalização do sistema numa pastilha de silício.

A tecnologia reconfigurável de circuitos integrados também pode ser utilizada diretamente em aplicações como sistemas embarcados, sem haver a necessidade da produção do chip em silício. Para o caso de desenvolvimento de uma simples solução, que não seja produzida em larga escala, recomenda-se o uso de dispositivos reconfiguráveis, tendo em vista que o custo de produção de um único chip pode se tornar inviável, tornando atrativa a utilização de circuitos reprogramáveis como alternativa para utilização do sistema.

Contudo, o desempenho de dispositivos reconfiguráveis pode ser lento quando comparado a dispositivos que efetuam aplicações específicas e executam apenas um conjunto limitado de instruções (HAUCK e DEHON, 2007). Nesta segunda modalidade de circuitos, é válido destacar os sistemas em hardware personalizado, também conhecidos como *Applications-Specific Integrated Circuits* (*ASICs*) – *Circuitos Integrados de Aplicações Específicas*. Devido a estas características específicas, os projetistas conseguem desenvolver chips cada vez menores, com desempenho melhor e que possuem um consumo energético menor que um processador de uso geral.

A principal característica da computação reconfigurável é a presença de um circuito em hardware que pode ser reconfigurado para implementar uma funcionalidade

específica mais apropriada e sob medida, e não um processador de propósito geral. Sistemas de computação reconfigurável unem os microprocessadores e o hardware programável com a finalidade de combinar o potencial do hardware e do software e ser utilizado em aplicações que vão desde um sistema embarcado a sistemas de alta performance computacional (ATHANAS e SILVERMAN, 1993), (OLUKOTUN, HELAIHEL, *et al.*, 1994).

A computação reconfigurável tem progredido extraordinariamente nas últimas duas décadas (ROSSI, 2012). Isto ocorre, principalmente, devido à larga aceitação dos dispositivos *FPGAs* que, cada vez mais, estão se estabelecendo como os dispositivos mais usados entre os que são reconfiguráveis.

2.2.1. *Field Programmable Gate Array* (*FPGA)*

A tecnologia *FPGA* (*Field Programmable Gate Array*) – *Arranjo de Portas Programável* em *Campo* – se trata de chips reprogramáveis (Figura 2.8) que contêm um número extenso (por volta de milhares) de unidades lógicas. Sua estrutura interna não é disposta das usuais portas dedicadas para a programação da função lógica (ROSSI, 2012), na verdade, cada uma de suas unidades lógicas, são pequenas matrizes de memória que são programadas com a função que se deseja a partir de uma tabela verdade (BERGER, 2002).

Dependendo da maneira como são implementados, alguns *FPGAs* podem ser reprogramados somente uma vez, enquanto outros podem ser diversas vezes reconfigurados (MAXFIELD, 2004). Um esboço físico de um *FPGA* pode ser visualizado na Figura 2.9.

Figura 2.8. Pastilhas de chip *FPGA*.

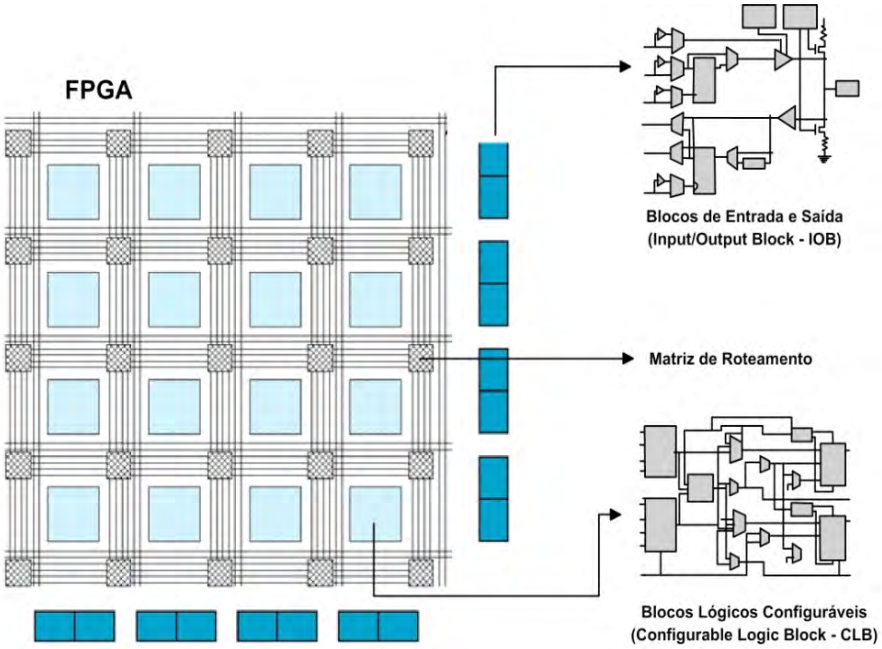


Figura 2.9. Esboço físico de um dispositivo *FPGA*.

Freqüentemente, dispositivos *FPGAs* são bastante utilizados em sistemas que operam o processamento de grandes fluxos de dados, tais como processamento de sinais e redes de computadores. Ao se comparar com dispositivos *ASICs*, podem ter um desempenho até 25 vezes inferior (HAUCK e DEHON, 2007). Contudo, projetos em *ASIC* são processos complexos que podem levar meses ou anos para serem concluídos, além de possuírem custos elevados, acrescentado a desvantagem de que o processo final em uma pastilha de silício é definitivo e não pode ser modificado sem a criação de um novo dispositivo (MAXFIELD, 2004). Ao utilizar dispositivos de tecnologia *FPGA*, é possível ter a vantagem da alteração da implementação do sistema desejado no "campo", ou seja, no local de trabalho onde está sendo desenvolvido o projeto do sistema, de uma forma bastante rápida e barata.

Apesar de comumente estarem presentes em placas de kit de prototipação de circuitos integrados, os chips *FPGAs* também podem ser adquiridos separadamente para serem acoplados a uma placa de configuração e programados para implementar o circuito desejado.

2.2.2. *PCI High-Speed Development Kit*

A forma mais comum de se iniciar o trabalho ou estudo de um tipo de *FPGA* é através do uso de kits de prototipação de projetos de circuitos integrados. Geralmente,

tais kits são placas que possuem, além do dispositivo *FPGA* em questão, interfaces de *E/S* para testes do sistema programado a ser embutido dentro da *FPGA*. A Figura 2.10 mostra como exemplo a imagem do kit de prototipação utilizado nas atividades deste trabalho: a placa *PCI High-Speed Development Kit*, desenvolvido pela a *ALTERA* ©.

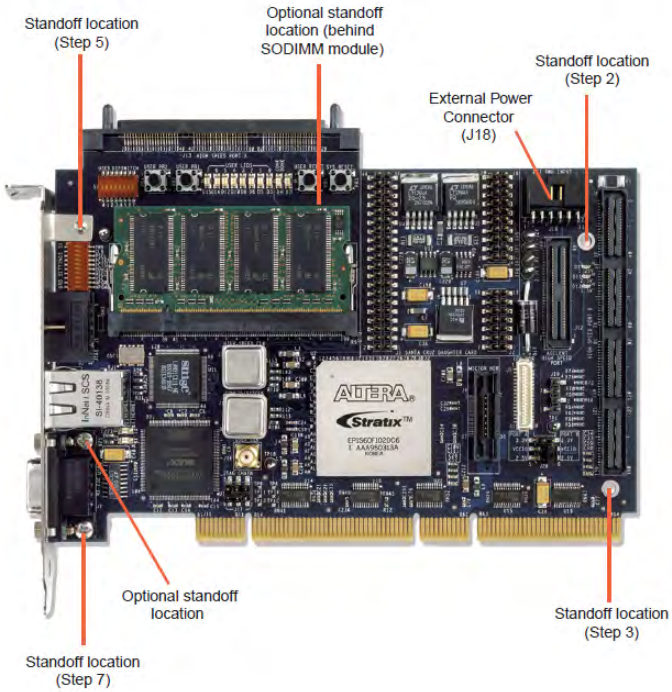


Figura 2.10. Kit de prototipação *Stratix PCI Development Board*.

A placa de desenvolvimento *Stratix PCI* é uma plataforma de avaliação e desenvolvimento para interfaces de alta velocidade incluindo *PCI*, *PCI-X*, *DDR SDRAM*, *Ethernet*, entre outras interfaces, com suporte a dispositivos *FPGA EP1S60F1020* da família *Stratix* (ALTERA, 2003).

O principal foco deste kit é o desenvolvimento de aplicações através da interface *PCI* ou *PCI-X*. O kit já possui um módulo de memória *DDR* de 256 MB incluída para uso nas aplicações utilizadas.

O dispositivo *Stratix* é conectado a todos os componentes através de interfaces adequadas *on-chip* e circuitos na placa, a Figura 2.11 abaixo exibe, em diagrama de blocos, as conexões estabelecidas pelo dispositivo *Stratix* no kit. O dispositivo suporta *PCI*, memória *DDR SDRAM*, e outras interfaces de alta velocidade. Os usuários podem programar o dispositivo *Stratix* para implementar seus sistemas lógicos. Sua programação é efetuada através da interface *USB* por meio do uso do cabo *ByteBlaster II*.

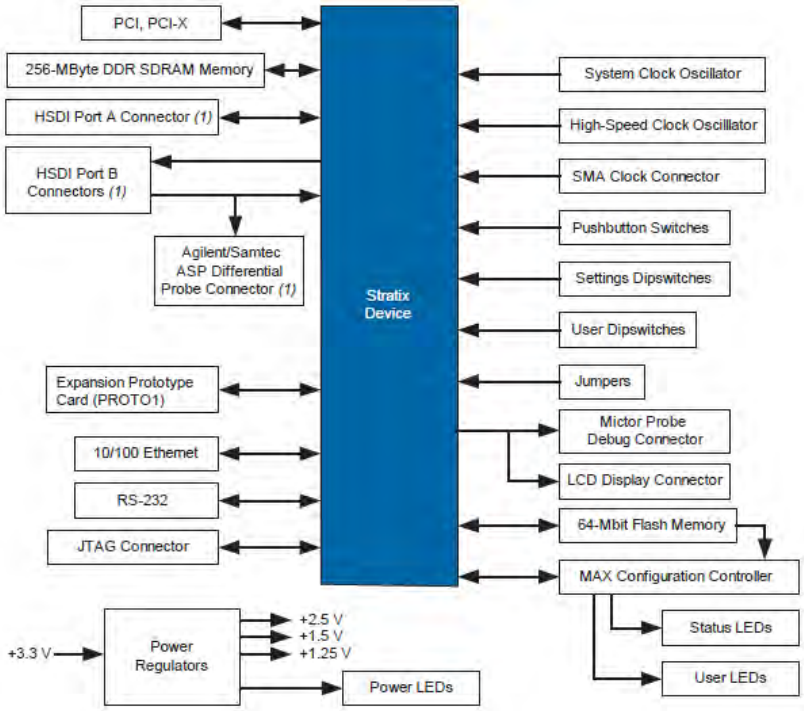


Figura 2.11 Esquema em blocos das conexões entre o dispositivo *Stratix* e as interfaces do *Kit Stratix PCI Development Board*.

O kit já vem com um projeto de referência embutido de fábrica, com possibilidade de customização, que efetua o controle da informação trocada entre o barramento *PCI* e a memória *DDR* presente no kit. Este projeto de referência encontra-se descrito em (ALTERA2, 2003). Basicamente o sistema efetua o controle de transações de dados em dois sentidos: do barramento *PCI* para a memória *DDR* do kit; da memória *DDR* do kit para o barramento *PCI*. A Figura 2.12 mostra o diagrama de blocos deste projeto de referência.

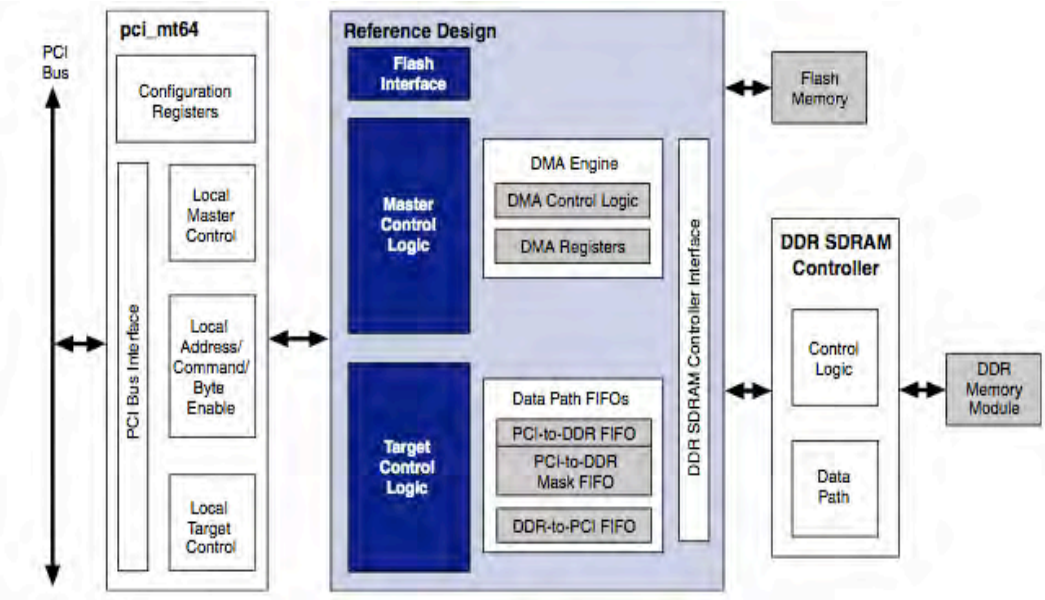


Figura 2.12 Projeto de referência de fábrica no kit de prototipação *Stratix PCI Development Board.*

2.3. Sistemas Embarcados

Uma das áreas em maior desenvolvimento tecnológico atualmente é a produção de computadores para auxiliar as atividades humanas, tanto no aspecto profissional quanto pessoal. São sistemas variados, em tamanho e forma, presentes nos mais diversos equipamentos em nossas casas ou locais de trabalho, que chegam a passar despercebidos quanto à sua condição de computador embarcado.

Há alguns anos era impossível imaginar a forte influência que os sistemas embarcados – também denominados de sistemas embutidos, sistemas dedicados, computadores embutidos, computadores dedicados ou computadores embarcados – exerceriam no nosso modo de vida. Seja em nossos momentos de lazer, em nosso ambiente de trabalho ou, ainda, nos meios em que nos comunicamos, tais sistemas aparecem em diversas aplicações, cada uma com características únicas.

Conforme um estudo desenvolvido por (TENNENHOUSE, 2000), somente 2% dos processadores produzidos são utilizados em computadores convencionais de uso geral (desktops e notebooks). O restante se encontra embarcado em dispositivos como aparelhos de telefonia móvel, eletrodomésticos, dispositivos de organização pessoal (*PDAs*), robôs, veículos e aeronaves. Os sistemas embarcados são utilizados largamente na indústria desde o surgimento dos primeiros microprocessadores, os quais fornecem

soluções digitais de baixo custo de produção e boa confiabilidade da integridade de seu uso (HEATH, 2003).

Por ser uma classe de dispositivos capazes de ter natureza de atuação bastante divergente, existem diversas formas para se classificar o que é um sistema embarcado. Entretanto, de acordo com (GANSSLE, 2008) a melhor maneira de defini-lo é de acordo com o propósito de seu uso, pois o seu desenvolvimento depende altamente da natureza do problema que o mesmo estará responsável por resolver dentro de outro sistema maior. Nesse ponto é que surge a denominação “sistema embarcado” que é a combinação de hardware e software, e algumas vezes peças mecânicas, desenvolvidos para realizar uma função específica (SILVA, 2006), ou seja, colocar capacidade computacional dentro de um circuito integrado, equipamento ou sistema.

2.3.1. Exemplificando Sistemas Embarcados

Como dito anteriormente, um sistema embarcado é a combinação de hardware e software, juntamente com a utilização ou não de peças mecânicas ou outras partes, projetados para efetuar uma função dedicada. Um bom exemplo que podemos citar são os aparelhos de reprodução de DVD, eletrodomésticos bastante comuns de se encontrar nas residências atualmente. Entretanto, poucas pessoas percebem que há uma arquitetura de hardware e software empenhada na reprodução dos filmes que reúnem a família em frente à televisão (BARR, 1999).

Isto, em parte, contrasta com a idéia do computador pessoal (*PC*) que possuímos em casa. Este é formado de hardware, software e equipamentos eletromecânicos tais como o disco rígido. Contudo, ele não foi projetado para desempenhar apenas uma atividade única, dedicada. Ao invés disso, é capaz de efetuar diversas tarefas diferentes. Muitas vezes utiliza-se o termo “computadores de propósito geral” para clarificar essa distinção. Ao se fabricar um computador de propósito geral, o fabricante não sabe o que o cliente vai fazer com ele. Uma pessoa pode utilizá-lo para produzir seus textos, se comunicar na internet e para o entretenimento através de jogos eletrônicos, enquanto outro usuário pode utilizá-lo como servidor de arquivos em uma rede ou para assistir seus filmes preferidos.

Em um sistema embarcado tem-se uma utilização bastante clara e dedicada à resolução de problemas específicos, que é uma alternativa de implementação do sistema através de uma arquitetura dedicada por meio de hardware e software, o que contribui de forma significativa com a otimização da velocidade do sistema em relação ao seu uso

em um computador de uso geral, por exemplo. Geralmente, estes sistemas possuem uma produção em larga escala e fazem parte de um sistema maior. Um bom exemplo são os diversos automóveis modernos que circulam nas ruas do mundo todo: diversos sistemas dedicados trabalhando concorrentemente para controlar a abertura do vidro da janela do carro, exibir a velocidade do automóvel ou até calcular a autonomia de consumo de combustível do veículo. Esses sistemas podem muitas vezes estar interligados entre si, formando, dessa forma, uma rede de sistemas embarcados diferenciados um dos outros.

2.3.2. Projetos de Sistemas Embarcados

Projetos de sistemas embarcados podem ser bastante complexos por necessitarem conhecimentos diversos. Muitas vezes, questões pouco exploradas ou ainda ignoradas em aplicações desktop são utilizadas nestes projetos, como por exemplo, a restrição ao consumo de energia e a pouca disponibilidade de memória. Um ponto crucial que pode afetar o projeto está relacionado ao espaço arquitetural, que torna a exploração das opções para a construção de um sistema ainda mais difícil (BABIERO, 2006), pois a arquitetura de hardware de um sistema embarcado pode conter um ou mais processadores, memórias de tipos variados, interfaces para periféricos e ainda blocos dedicados.

Devido às variações nas camadas de hardware de um sistema embarcado, estes não podem ser transportados para outros dispositivos e serem executados normalmente sem que haja modificações significativas em seus projetos em nível de hardware.

Em cada sistema embarcado, os projetos de hardware são adaptados de forma que os custos em seu desenvolvimento sejam os mínimos possíveis, eliminando, dessa forma, qualquer estrutura desnecessária. De acordo com (BARR, 1999), um sistema embarcado possui por definição um processador e um programa para sua execução. Contudo, o estudo de algumas características importantes pode ser levado em consideração durante o projeto do sistema embarcado, tais como o tempo de desenvolvimento, o custo e o tempo de vida. Caso seja necessário um software para o processamento das atividades do sistema, é preciso um local de armazenamento do código do programa executável e dos dados de entrada e saída manipulados durante os cálculos do sistema. Tal armazenamento se dá por meio de unidades de memórias acopladas ao sistema embarcado.

O desenvolvimento de algumas partes dos sistemas diretamente em circuito favorece um desempenho mais alto de execução em relação à implementação das

mesmas partes em software. Contudo, o desenvolvimento do projeto em hardware é mais custoso, requerendo também um maior tempo para seu desenvolvimento.

Apesar do desenvolvimento em software utilizando microprocessadores apresentar, além de uma maior flexibilidade, um menor custo e requerer menos tempo para o seu desenvolvimento, seu resultado possui desempenho baixo em relação à implementação em hardware. Em alguns casos, para se desenvolver um determinado sistema, o ideal é que seja possível obter desempenho ótimo no resultado final, porém o desenvolvimento também deve ser efetuado mediante uma estratégia flexível e rápida. De acordo com (BONATO, MAZZOTI e MARQUES, 2009 apud GAMBARRA, 2010), uma solução para esse problema é o desenvolvimento conjunto “hardware x software”, onde algumas partes de um sistema são desenvolvidas em hardware, e outras efetuadas em software. Como resultado final dessa abordagem é possível ter um aumento do desempenho em relação a um desenvolvimento efetuado em sua maior parte em software, com uma maior flexibilidade.

Seguindo esta abordagem, operações com desempenho crítico e que possuem o potencial de atuarem em paralelo devem ter sua implementação desenvolvida em hardware, enquanto que as operações seqüenciais e mais complexas podem ter sua implementação na forma de software. Conforme descrito em (BONATO, MAZZOTI e MARQUES, 2009 apud GAMBARRA, 2010), a Figura 2.13 ilustra a relação existente entre o custo, a flexibilidade e o desempenho no desenvolvimento de sistemas embarcados em software, hardware, ou desenvolvimento conjunto de software e hardware.

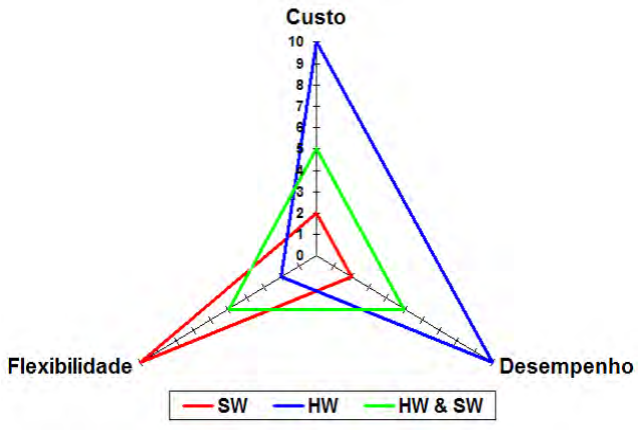


Figura 2.13. Relação entre custo, flexibilidade e desempenho.

Além disso, é necessário que todos os sistemas também possuam algum tipo de entrada e saída. Por exemplo, no caso citado do aparelho de reprodução de DVD, o controle remoto serve como a interface de entrada de dados, enquanto a tela e caixas de som são periféricos que, acoplados ao aparelho de DVD, demonstram os resultados de saída processados: a imagem e o som do filme. A Figura 2.14 mostra o modelo básico de um sistema embarcado.

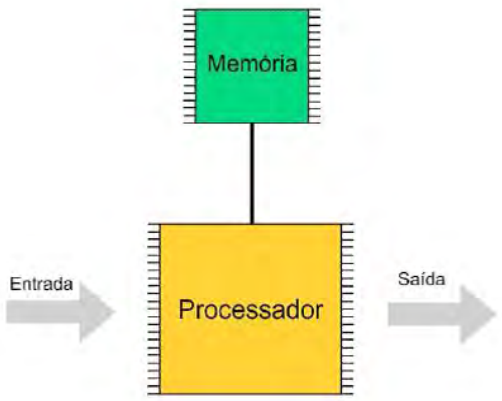


Figura 2.14. Modelo básico de um sistema embarcado.

2.4. Paralelismo

Ao longo dos anos, é possível observar os esforços dos projetistas de computadores para se desenvolver máquinas com desempenho cada vez mais potentes. É visível o rápido avanço tecnológico que se pode ter em espaços de meses. Fabricar circuitos integrados que executem numa maior freqüência de operação do sinal de *clock* é possível, porém sempre há limitações para as tecnologias de cada época (TANENBAUM, 2007). Contornando os empecilhos, a idéia de sistemas executando partes paralelamente surge como a alternativa para se conseguir resultados mais rápidos para uma dada freqüência de sinal de *clock*.

As pesquisas atuais, em arquitetura de processadores modernos, baseiam-se principalmente na capacidade de execução de várias operações em um único ciclo de sinal de *clock*, implicando no paralelismo das operações envolvidas. Usualmente, as técnicas de execução em *pipeline* e execução superescalar são as mais utilizadas em abordagens de paralelismo de sistemas computacionais.

A execução das operações utilizando uma abordagem em *pipeline* segue a idéia em que cada operação é executada em várias etapas, sendo uma sobreposição de várias

operações menores que a compõem, tal qual a linha de montagem de uma fábrica industrial. As atividades são executadas paralelamente em fases diferenciadas. Dessa forma, o *pipeline* pode ser definido em hardware através da execução paralela de um certo número de blocos de circuitos dedicados.

A Figura 2.15 exibe a estrutura de um *pipeline* de cinco estágios para o tratamento de uma instrução de um determinado processador. A parte (*a*) da imagem exibe os blocos componentes dos estágios da operação de tratamento da instrução. A parte (*b*) mostra a superposição de atividades entre os blocos ao longo dos ciclos de *clock* com relação à instrução sendo tratada no momento.

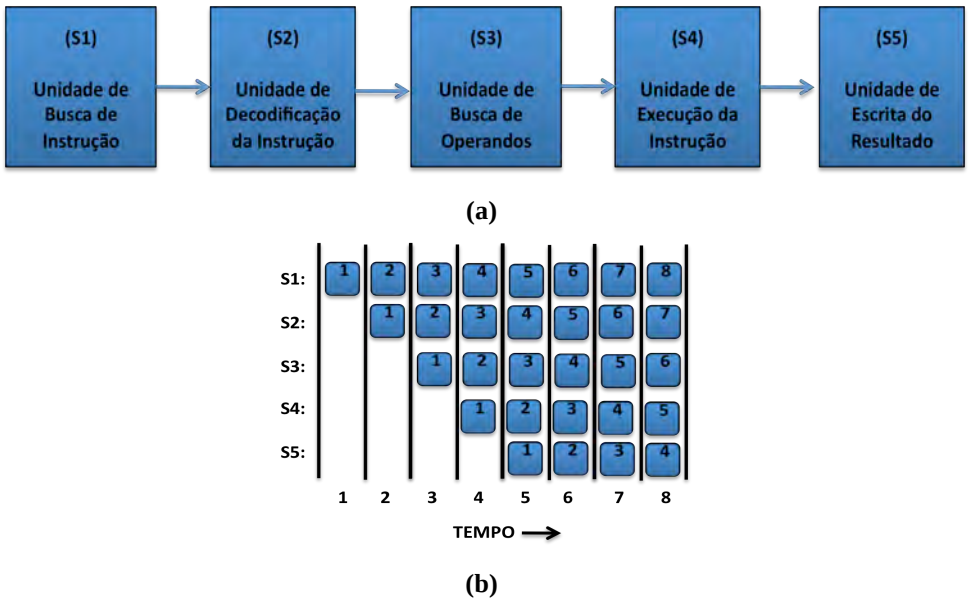


Figura 2.15. Exemplo de *pipeline* de cinco estágios.

Na execução superescalar as operações são executadas em paralelo, exigindo a multiplicação dos blocos funcionais para atingir o número de operações paralelas desejadas. Neste tipo de arquitetura, todas as operações são executadas em um único ciclo de sinal de *clock* (TANENBAUM, 2007).

Na Figura 2.16 há a demonstração de execução superescalar por meio de *pipelines* duplos para efetuar a mesma atividade representada pela Figura 2.15, porém de forma mais rápida devido ao paralelismo dos blocos.

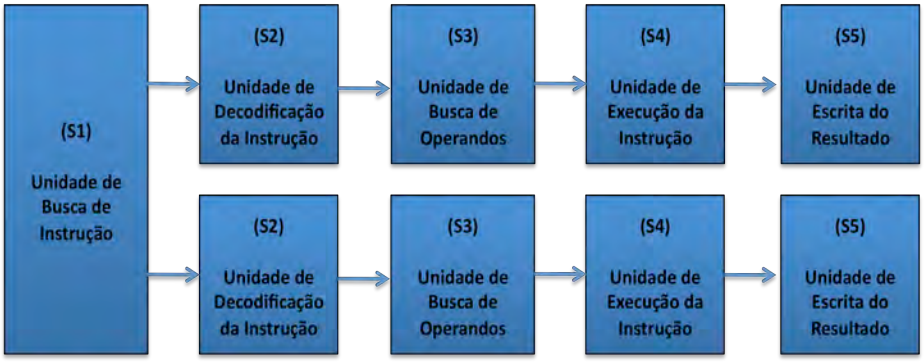


Figura 2.16. Paralelismo através da execução superescalar de operações.

A utilização das abordagens de paralelismo exemplificadas permitem reduzir o tempo de execução de operações no hardware. O *pipeline* desmembra uma operação em várias, o que diminui o período de ciclo de *clock*, em contrapartida a operação leva vários ciclos para ser concluída. A execução superescalar permite o paralelismo de várias operações ao preço de aumentar a área utilizada do circuito do sistema.

2.5. Protocolo *AMBA AXI*

O protocolo *AMBA* (*Advanced Microcontroller Bus Architecture*) é um padrão aberto de especificação de barramentos *on-chip*, fornecido pela empresa *ARM*, escrito para auxiliar engenheiros de software e hardware numa estratégia para interconexão e gerenciamento de blocos funcionais que compõem um sistema em *chip* (ARM, 2012). A utilização do AMBA faz com que o reuso de sistemas em chip seja permitido devido ao seu poder de utilização genérica para o desenvolvimento de módulos em chip.

As vantagens de utilização do padrão AMBA, que vão desde a fácil adaptação para reuso de blocos até a possibilidade de um projeto multicamadas (separação de responsabilidades e interconexão de vários módulos dentro do sistema), são bastante visíveis na indústria da microeletrônica. Apesar disso, o protocolo foi estendido posteriormente, devido às necessidades do mercado, com a inclusão de outros protocolos, dentre eles, o *AXI* (*Advanced Extensible Interface*) (ARM, 2012).

2.5.1. Arquitetura

O protocolo é baseado em rajadas. Cada transação tem endereço e informação de controle no canal de endereço que descreve a natureza do dado a ser transferido. Há duas possibilidades de transferências de dados: O dado é transferido do dispositivo

mestre para o escravo utilizando um canal de escrita de dados ou o dado é transferido no sentido do escravo para o mestre, utilizando um canal de leitura de dados.

Cada canal da arquitetura *AMBA AXI* consiste de um conjunto de sinais que utiliza um mecanismo de *handshake* utilizando os sinais *READY* e *VALID.* O sinal *VALID* é utilizado pelo módulo fonte da informação para indicar quando um dado válido ou informação de controle está disponível no canal. O módulo destino usa o sinal *READY* para indicar que pode aceitar o dado. *LAST* indica quando a transferência do último dado da transação vai ocorrer.

O padrão, atualmente, descreve a utilização de cinco canais de *handshake* para a transferência de informação:

- Canal de escrita de endereço;
- Canal de escrita de dado;
- Canal de resposta de escrita;
- Canal de leitura de endereço;
- Canal de leitura de dado.

2.5.2. Transações Básicas

O padrão *AMBA AXI* permite dois tipos de transações: a leitura de rajada de dados e a escrita de rajada de dados. As transferências ocorrem quando há ativação de ambos os sinais *READY* e *VALID.* A Figura 2.17 exibe a rajada de quatro transferências de leitura.

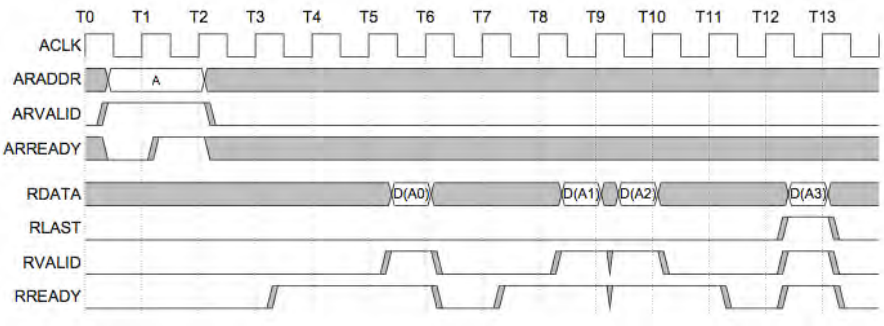


Figura 2.17. Rajada de leitura do *AMBA AXI.*

Na figura, o sinal *ACLK* refere-se ao pulso do *clock* do sistema. Os sinais iniciados por *AR* (*ARADDR, ARVALID* e *ARREADY*) referem-se aos sinais do canal de

endereço de leitura. Por fim, os sinais iniciados por *R* (*RDATA, RLAST, RVALID* e *RREADY*) referem-se aos sinais do canal de leitura de dados.

No exemplo da Figura 2.17, após o endereço aparecer no barramento de endereços, a transferência de dados ocorre no canal de leitura de dados. O escravo mantém o sinal *VALID* desativado até que dado para leitura esteja disponível. Para a última transferência da rajada, o escravo ativa *RLAST* para mostrar que o último dado está sendo transferido.

Para exemplo de transações de escrita, tem-se a Figura 2.18.

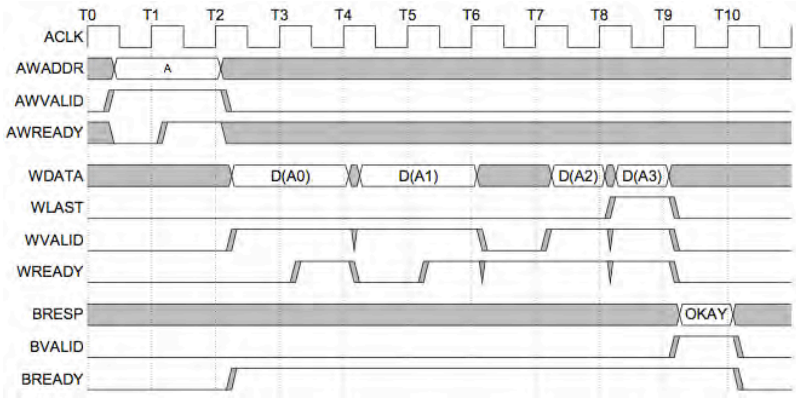


Figura 2.18. Rajada de escrita do *AMBA AXI*.

No exemplo da Figura 2.18, o processo inicia quando o mestre envia um endereço e informação de controle no canal de endereço de escrita (sinais *AWADDR, AWVALID* e *AWREADY*). O dispositivo mestre então envia cada item da escrita de dados sobre o canal de escrita de dados (sinais *WDATA, WLAST, WVALID* e *WREADY*). Quando o mestre envia o último item de escrita, o sinal *WLAST* é ativado. Ao aceitar todos dados do mestre, o escravo envia uma resposta ao mestre (sinais *BRESP, BVALID* e *BREADY*) para indicar que a transação está completa.

2.5.3. *Handshake* entre Canais

Todos os cinco canais usam os mesmos sinais *VALID/READY* para efetuar o *handshake* de transferência de informação. O módulo fonte gera o sinal de *VALID* e o destino gera o *READY*. A transferência ocorre apenas quando ambos *VALID* e *READY* estão ativados.

Para o canal de escrita de endereço, o módulo mestre ativa o sinal *AWVALID* quando libera endereço válido e informação de controle. *AWVALID* permanece ativo até

que o módulo escravo aceite o endereço e informação de controle e, por fim, ativa o sinal *AWREADY*.

Já no canal de escrita de dado, o módulo mestre ativa o sinal *WVALID* apenas quando libera dado de escrita válido. O sinal *WVALID* deve permanecer ativado até que o escravo aceite o dado de escrita e o sinal *WREADY*. *WLAST* é ativado quando o mestre libera a última transferência da rajada.

O canal de resposta de escrita é inicializado pelo módulo escravo. O escravo ativa o sinal *BVALID* quando libera uma resposta de escrita válida. *BVALID* fica ativo até o mestre aceitar a resposta e ativar *BREADY*.

Para usar o canal de leitura de endereço, o mestre ativa o sinal *ARVALID* quando libera endereço válido e informação de controle. Deve permanecer ativo até o escravo aceitar o endereço e informação de controle e ativar *ARREADY*.

Caso o módulo escravo possua dado de leitura válido, aciona o canal de leitura de dado. O escravo ativa o sinal *RVALID*, este permanece ativado até que o módulo mestre ative *RREADY*, indicando que aceita dado. *RLAST* deve ser ativado quando a última transferência da rajada deve ser liberada.

As Figuras 2.19, 2.20 e 2.21 exibem exemplos de seqüências de *handshake* generalizados. Não importa o canal que efetua o *handshake*, os resultados para qualquer canal seguem os exemplos das figuras.

Na Figura 2.19, o módulo fonte libera o dado ou informação de controle e ativa o sinal de *VALID*. A informação permanece estabilizada até que o módulo destino ative o sinal *READY*, indicando que aceita a informação.

Já na Figura 2.20, o módulo destino ativa o sinal *READY* antes da informação ser válida. Isso indica que o módulo destino pode aceitar a informação em um único ciclo de *clock* assim que o *VALID* for ativado.

Por fim, na Figura 2.21, tanto o módulo fonte quanto o destino indicam no mesmo ciclo de *clock* a permissão da transferência da informação. Neste caso, a transferência ocorre imediatamente.

Em todas as figuras, a seta indica exatamente quando a transferência ocorre.

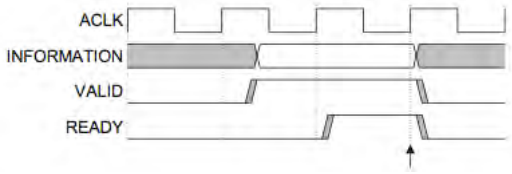


Figura 2.19. *Handshake* exibindo o sinal *VALID* ativado antes do sinal *READY.*

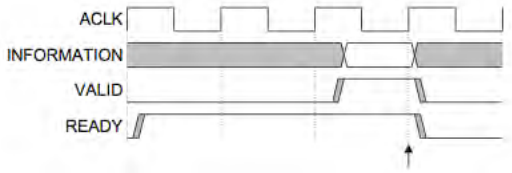


Figura 2.20. *Handshake* exibindo o sinal *VALID* ativado depois do sinal *READY.*

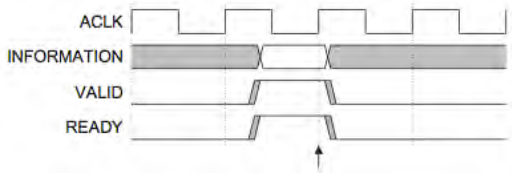


Figura 2.21. *Handshake* exibindo o sinal *VALID* ativado ao mesmo tempo do sinal *READY.*

III. Sistema Misto Reconfigurável Aplicado à Interface PCI para Otimização do Algoritmo *Non-local Means*

3.1. Desenvolvimento Misto "Software x Hardware"

Para o desenvolvimento da arquitetura do sistema misto foi necessária a investigação do algoritmo *NLM* para se decidir quais partes seriam contempladas em software e quais seriam definidas em hardware.

Seguindo as idéias presentes em (GAMBARRA, 2010), que unem alto desempenho e flexibilidade, foi decidida a implementação em hardware do cálculo da *Distância* L^2 e a implementação em software do restante do algoritmo.

Para clarificar, a Figura 3.1 mostra o fluxo de processamento do algoritmo *NLM* em software utilizando a abordagem da otimização por janelas de busca, juntamente com as desejadas otimizações em hardware destacadas. O sistema está utilizando janelas de busca de tamanho 21x21, *kernel gaussiano* de tamanho 7x7 e imagens em escala de cinza, para seguir o recomendado em (BUADES, COLL e MOREL, 2004).

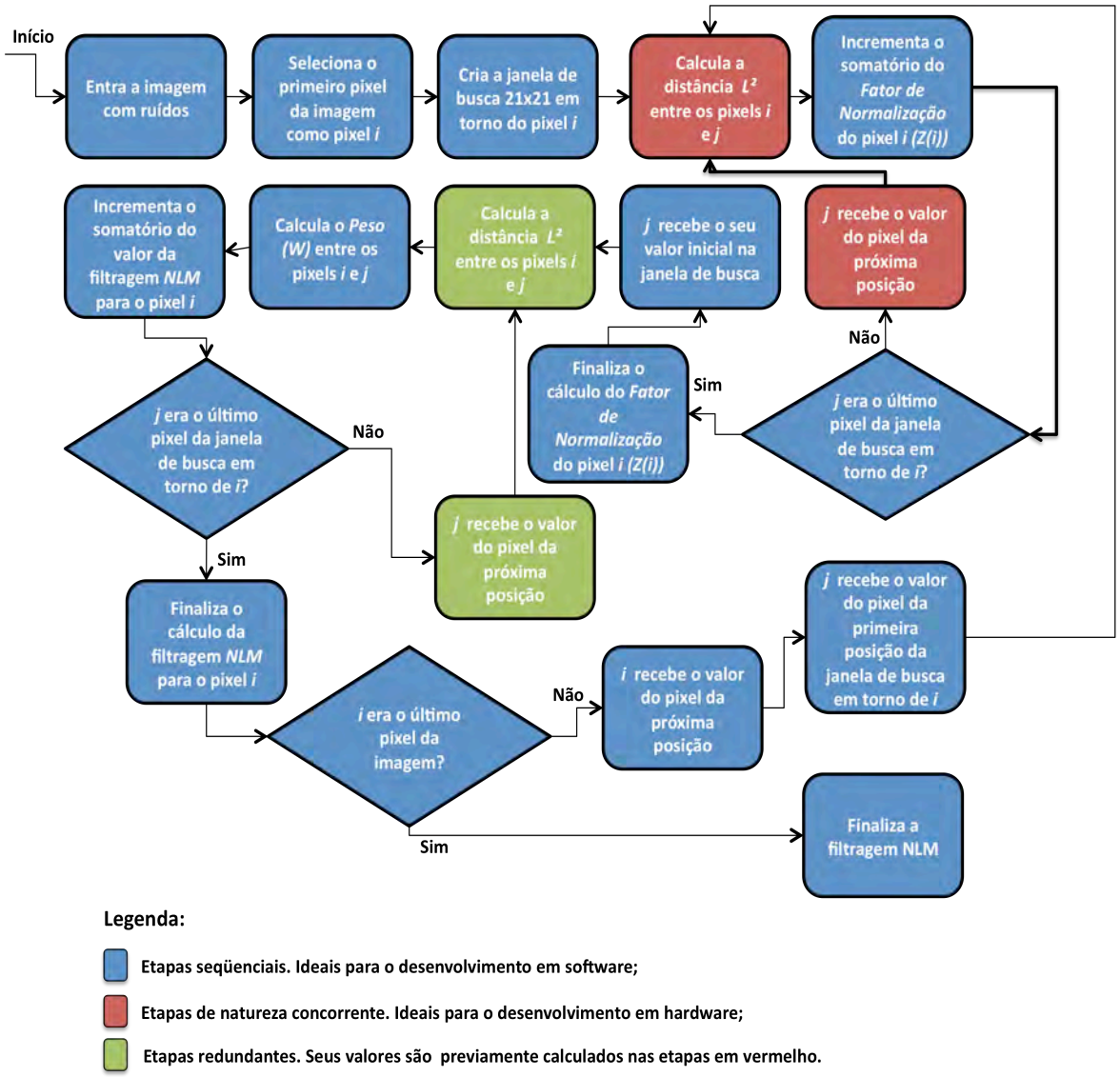


Figura 3.1. Fluxo do processamento do algoritmo *NLM*.

3.2. Paralelização do Cálculo da Distância Euclidiana Quadrática Ponderada

Na configuração do *NLM* investigada em software está sendo utilizada uma janela de vizinhança 7x7, o que culmina em 49 operações aritméticas para se concluir o cálculo da *Distância* L^2 sobre os vetores *i* e *j*. Porém, em software estas 49 operações independentes ocorrem sequencialmente, enquanto a natureza do problema permite o cálculo concorrente de cada uma destas operações.

Percebe-se também que este cálculo estará restrito à janela de busca de tamanho 21x21 em torno do pixel *i*. Assim, todos os 441 pixels independentes, internos a esta janela de busca, em algum momento no sistema seqüencial do software, serão o pixel *j* referenciado. Portanto, cada um dos pixels terá um valor independente do cálculo da

Distância L^2 com o pixel *i*. Neste ponto do processamento do algoritmo, também é possível enxergar a natureza concorrente dos cálculos. Contudo, seriam 441 x 49 operações concorrentes, o que resulta em 21609 operações de cálculo da *Distância* L^2 acontecendo ao mesmo tempo.

Assumindo o fato de que é dispendioso o cálculo das 21609 operações supracitadas, para que o cálculo integral de uma janela de busca fosse efetuado é necessário o cálculo parcial da mesma, o que resulta em um determinado número de iterações a serem concretizadas. Finalizados tais cálculos, o algoritmo pode seguir em frente: o hardware passaria para o software um vetor contendo todos os valores dos cálculos da *Distância* L^2, referente à janela de busca daquele momento, o sistema prosseguiria com o cálculo e finalização do fator de normalização do pixel *i* (*Z(i)*), enquanto o hardware poderia ir adiantando a próxima janela de busca.

Mediante as observações citadas acima, é possível reajustar o fluxo do algoritmo para a configuração exibida na Figura 3.2.

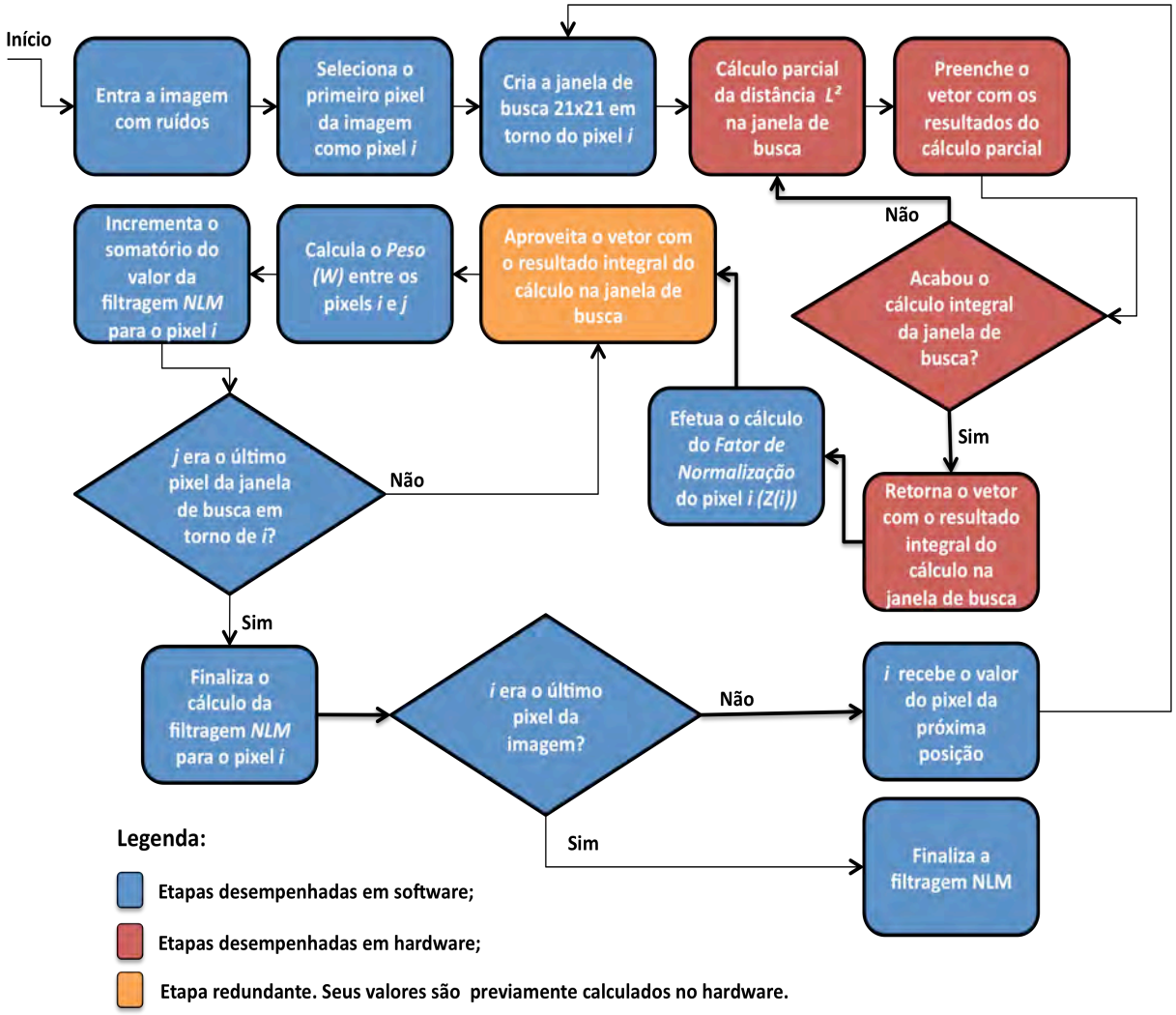


Figura 3.2. Fluxo do processamento do algoritmo *NLM* com o desenvolvimento do sistema misto.

3.3. Arquitetura Proposta

Observando as considerações que foram expostas acerca do novo fluxo de processamento de filtragem do *NLM*, é possível idealizar uma arquitetura para o circuito que será descrito no *FPGA*.

Inicialmente, baseando-se num simples circuito para o fluxo acima, bastaria a implementação de um sistema em hardware que:

- Receba como entrada a janela de busca 21x21, juntamente com o vetor *N(i)*;
- Efetue os cálculos relacionados à *Distância* L^2;
- Libere na saída um vetor com o resultado do cálculo da *Distância* L^2 entre o vetor

N(i) e o *N(j)* da vez.

A Figura 3.3 demonstra sob a forma de diagrama de bloco um esquema do módulo *Distância L2*, que efetuaria as atividades descritas.

Figura 3.3. Módulo do cálculo da *Distância L2*.

De posse da proposta do módulo para o cálculo da *Distância* L^2, é preciso incorporá-la ao sistema do kit de prototipação *FPGA* onde é efetuada sua descrição em determinada linguagem de descrição de hardware. Conforme citado anteriormente, o kit de prototipação *Stratix PCI Development Board*, entre outras utilidades oferecidas, permite o processamento desejado por meio da interface *PCI* do computador *Host* com o auxílio do módulo *DDR* de memória já presente no kit, que serve como a parte do armazenamento das informações durante o cálculo da *Distância* L^2.

Dessa forma, o sistema *NLM*, atuando com o hardware embarcado da *Distância* L^2, atuará no seguinte fluxo:

- O software recebe a imagem, inicialmente ruidosa;
- Seleciona a janela de busca juntamente com o vetor *N(i)* desejado;
- Através da interface *PCI* a janela de busca e o vetor *N(i)* são enviados ao módulo de memória presente no kit;
- O módulo responsável pelo cálculo da *Distância* L^2 efetua a requisição dos valores da memória para iniciar o cálculo;
- Após o cálculo parcial ter sido finalizado, o módulo responsável pelo cálculo da *Distância* L^2 vai incrementando o vetor que estará na memória *DDR* do kit;
- Se houver cálculos parciais ainda a serem feitos, o módulo faz nova requisição à memória;
- Caso não haja mais cálculos a serem feitos, o hardware sinaliza para o software, este faz a requisição dos dados do vetor das *Distâncias* L^2 na memória do kit e o

hardware aguarda o envio, por meio do software, da próxima janela de busca e vetor *N(i)* para iniciar novos cálculos.

Abaixo, demonstrada na Figura 3.4, temos um esboço alto-nível da arquitetura em diagrama de blocos do sistema completo em suas partes de hardware e software. O bloco de *Lógica de Controle Interno* é o módulo que fará o controle dos sinais trocados entre a *Lógica de Controle da Interface PCI,* a *Lógica de Controle da Memória DDR* e o módulo do cálculo da *Distância* L^2.

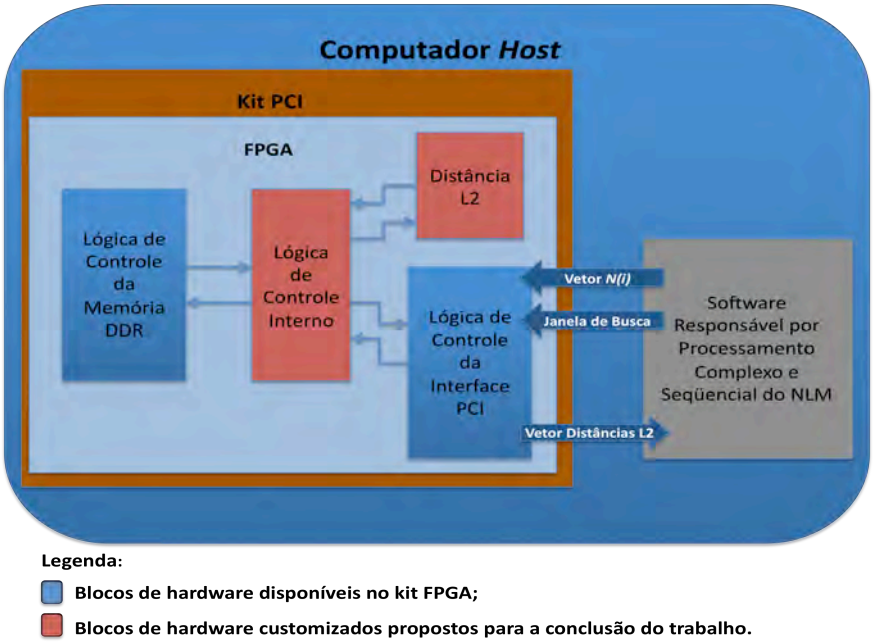


Figura 3.4. Diagrama de blocos da arquitetura do sistema proposto.

Para efetuar o processamento do bloco *Distância L2*, este é dividido em sub-blocos internos. A divisão permite que se concentre pequenas quantidades de cálculos em cada módulo a ser dividido, facilitando o reuso do sistema, e também agiliza o processo do cálculo ao utilizar uma abordagem com *pipelines* entre os sub-blocos.

Segundo (GAMBARRA, 2010), a divisão do *Distância L2* implicou nos cinco blocos internos:

- Subtração: responsável pelo cálculo do termo $(x_m - y_m)$ da Equação 2.7;

- Quadrado: eleva ao quadrado o resultado efetuado pelo módulo *Subtração*;
- Soma I: soma os resultados do módulo *Quadrado* que possuem termos a_m de mesmo valor;
- Pondera: efetua a multiplicação entre os resultados do módulo *Soma I* e os termos a_m correspondentes;
- Soma II: cálculo final do somatório dos valores que saem do módulo *Pondera*.

Para completar a arquitetura dos sub-blocos internos, a abordagem pretendida adicionou mais dois blocos:

- Vetor NxNi: mantém os valores do vetor *N(i)* a ser tratado;
- Janela de Busca: mantém os valores da janela de busca a ser tratada.

Dessa forma, é possível exibir o esquema de conexão entre os sub-blocos internos do *Distância L2* através da Figura 3.5. A comunicação entre os blocos obedece aos sinais dos canais do protocolo de *handshake AMBA AXI.*

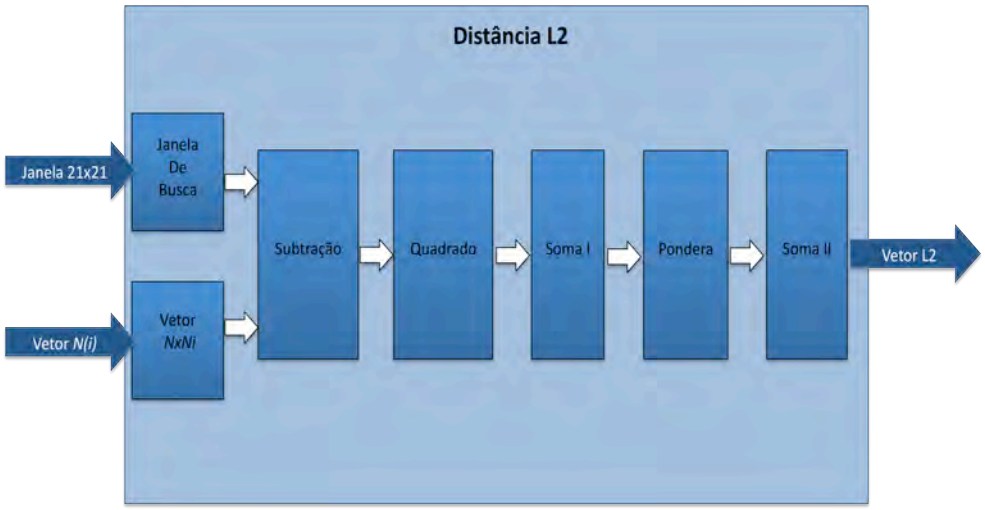


Figura 3.5. Arquitetura interna do bloco *Distância L2*.

3.4. Resumo do Sistema

O sistema idealizado será composto de partes em software e hardware para a otimização do tempo de conclusão do *NLM*.

O software, que estará presente no computador *host*, deverá ficar encarregado de processar partes do algoritmo que são de natureza seqüencial ou que possuem cálculos complexos para serem desenvolvidos em hardware. O mesmo deverá efetuar suas atividades e se comunicar com o hardware especializado para este continuar com o

processamento do filtro.

O hardware especializado está embutido no kit *FPGA* e sua comunicação com o software se dará mediante a interface *PCI* do computador *host*. As atividades que o hardware efetuará são as partes do processamento do algoritmo *NLM* que possuem natureza de processamento paralelo para, assim, otimizar o tempo de conclusão do processo de filtragem.

A Figura 3.6 exibe um esquema da visão do sistema.

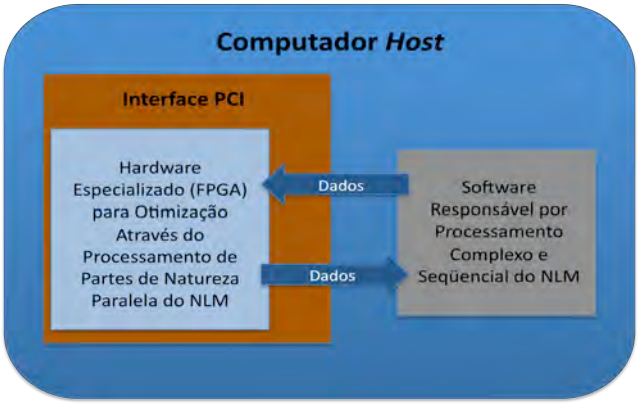


Figura 3.6. Esquema da visão do sistema de otimização do algoritmo *NLM*.

IV. Materiais e Métodos

Estabelecida a estrutura do projeto do sistema, nesta seção serão tratados os métodos e ferramentas empregadas no desenvolvimento das partes do sistema em software e hardware.

4.1. Camada de Software

A camada de software do sistema desenvolvido engloba tanto a parte da execução do *NLM* em software, responsável pelo já esclarecido processamento seqüencial, quanto o *driver* de comunicação com a interface *PCI* do computador *host* onde foi instalada a placa de prototipação *FPGA*.

A linguagem de programação *C*, obedecendo ao padrão *C99*, foi selecionada para quaisquer implementações de códigos do algoritmo. O ambiente de programação utilizado é o *Code::Blocks* v10.05 compatível com sistemas operacionais *Windows*, executando o compilador *GNU GCC 4.6.0*. Uma máquina *Intel Core 2 Duo (3 MB Cache, 2.4 GHz), Windows XP SP2 - 32 Bits* foi a selecionada para os desenvolvimentos.

4.1.1. Algoritmo *NLM* em Software

Para início da pesquisa foi necessária a implementação integral do algoritmo *NLM* em software. A abordagem obedece à implementação otimizada das janelas de busca que obedece àquela exemplificada na Figura 2.6 apresentada anteriormente (vide página 24). Esta implementação obedece àquela descrita em (BUADES, COLL e MOREL, 2004), já intensamente testada, obedecendo aos requisitos da filtragem *NLM*.

4.1.2. Driver de Comunicação com Interface *PCI*

Para haver a comunicação esperada entre as partes de hardware e software do sistema é necessário o desenvolvimento do *driver* que fará o controle das informações passadas entre tais partes. O *driver* deverá se comunicar com os sinais lançados através da interface *PCI* do computador *host* devido ao kit de prototipação *FPGA* estar conectado a tal interface, como explicado anteriormente.

Para o desenvolvimento do *driver* de comunicação da placa de prototipação foi utilizada a ferramenta *WinDriver* que vem disponibilizada juntamente com o pacote do kit *FPGA*. O *WinDriver* é um kit de desenvolvimento que simplifica a tarefa de se criar aplicações de acesso a hardware. *WinDriver* inclui um software guia com facilidade de geração de código que automaticamente detecta o hardware e gera o driver para acessá-lo da aplicação em software (JUNGO, 2010).

A Figura 4.1 exibe a arquitetura de sistema do driver gerado com o auxílio do *WinDriver* para o sistema proposto.

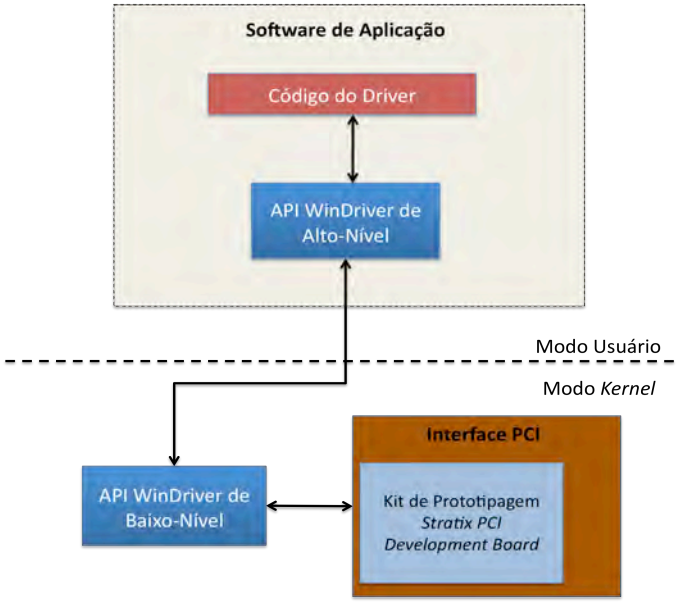


Figura 4.1. Arquitetura do *driver* para o kit *Stratix PCI Development Board.*

A comunicação com a placa *PCI* de prototipação *FPGA* é possível devido ao projeto de referência embutido de fábrica na *FPGA* do kit descrito em (ALTERA2, 2003), exemplificado na seção 2.2.2 deste documento.

O driver gerado através do uso do *WinDriver* é descrito em linguagem de programação *C*. O software de aplicação gerado para exemplificar o uso do driver

efetua transações de pacotes de bytes entre o computador *host* e a memória *DDR* do kit. O código da aplicação foi customizado de modo que o usuário envia um arquivo interno do computador *host* para a memória do kit e efetua o resgate de tal dado da memória do kit para o computador *host.*

4.1.3. Variável Compartilhada

Para a correta interação entre o software e o hardware foi preciso criar uma variável compartilhada de controle de operação. Esta variável será o elo do processamento misto, através dela o software e o hardware sinalizarão um ao outro para efetuarem o processamento através de códigos para cada operação.

Uma variável de 8 bytes foi selecionada para ser a variável compartilhada. Inicialmente, ela possuirá apenas dois códigos de operação:

- (0x00000000 00000000): Código de leitura da memória. Indica que o hardware já processou o dado enviado e que, deste modo, o software pode iniciar uma leitura do dado completo na memória do kit;
- (0x00000000 00000001): Código de processamento em hardware. Indica ao hardware que o dado a ser processado está presente na memória do kit e que, deste modo, o processamento poderá ser iniciado.

4.2. Camada de Hardware

Como será necessário o uso da memória *DDR* presente no kit de prototipação *FPGA* para armazenamento dos dados que serão passados via software, o sistema será baseado no projeto de referência descrito em (ALTERA2, 2003) para o desenvolvimento da parte do *NLM* em hardware, que será responsável pelo cálculo da *Distância* L^2.

Para integralizar o estudo, utilizou-se o ambiente *Quartus II v8.0 SJ Full Version* da *ALTERA©* para a descrição em hardware dos blocos necessários. Estes foram descritos em linguagem de descrição de hardware *VHDL* seguindo o padrão definido em *IEEE Std 1076.3-1997*. A escolha do *VHDL* foi efetuada para aproveitar o código já descrito do projeto de referência.

4.2.1. Customização do Projeto de Referência

Para a inclusão do módulo que efetua o cálculo da *Distância* L^2, é preciso

customizar a lógica de controle interno do sistema de referência, ou seja, o projeto embutido de fábrica.

A customização do hardware descrito em (ALTERA2, 2003) é necessária pois o mesmo apenas efetua transações de escrita e leitura na memória *DDR* do kit *FPGA*, nada mais que isso. Para a aplicação proposta, além de efetuar as transações descritas, é necessário o processamento dos dados que chegam na memória do kit.

Dessa forma, foi necessário o desenvolvimento de uma lógica interna para o controle do processamento dos dados da memória do kit, cujo o fluxo é o seguinte:

- Os dados serão enviados via software para a memória *DDR* do kit;
- Após o dado estar completamente na memória, a lógica customizada deverá acessá-lo, efetuar seu devido processamento e retornar seu resultado novamente na memória;
- Por fim, o trecho de memória com os resultados é passado para o barramento *PCI*.

A Figura 4.2 mostra o diagrama de blocos da idealização do sistema customizado.

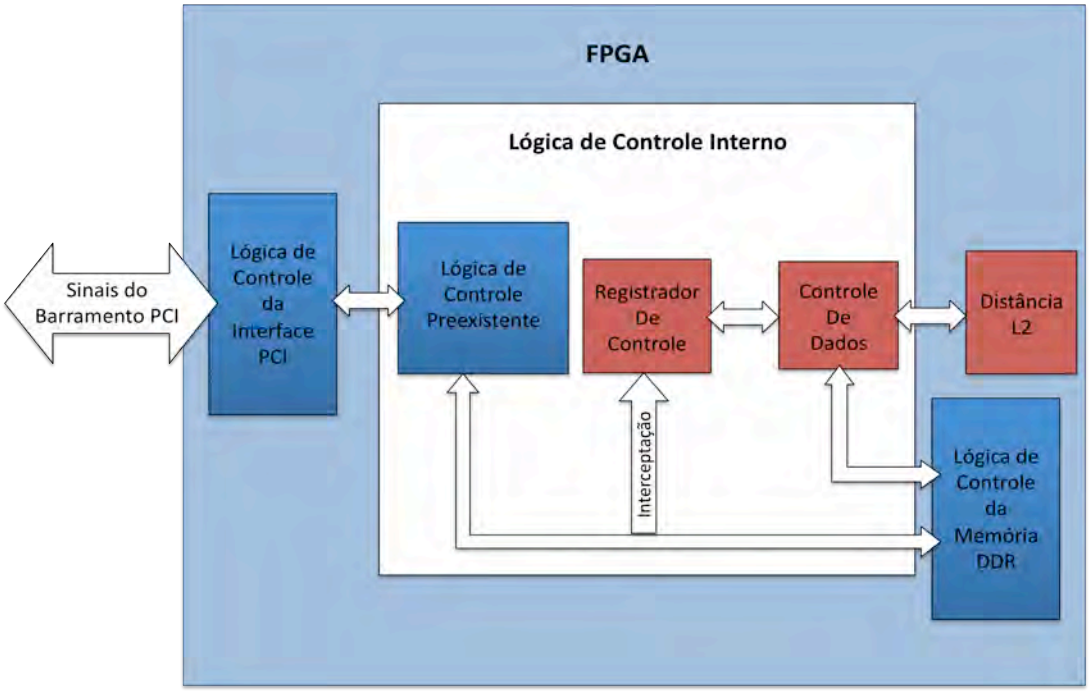


Figura 4.2. Diagrama de blocos da customização projeto de referência do kit *FPGA*.

Inicialmente, o dado será inserido no sistema por meio dos sinais do barramento *PCI* através da lógica de controle da interface *PCI* e, em seguida, será tratado pela lógica de controle interno.

Inserido na lógica de controle interno, o dado é passado à lógica de controle preexistente do projeto de referência. Essa lógica efetua o controle das operações de *E/S* de dados da memória *DDR* do kit de prototipação, pois atua diretamente sobre os sinais da lógica de controle da memória. Assim, conclui-se o ciclo completo do sistema original do kit, representado pelos blocos azuis e suas interconexões, onde as operações efetuadas pelo sistema são simples transações de *E/S* de dados da memória *DDR* do kit. Contudo, deseja-se efetuar o cálculo, já amplamente discutido em todo este texto, da *Distância* L^2 e para tanto se fazem necessárias novas lógicas a serem inseridas no sistema, que são aquelas representadas em laranja e suas interconexões.

O bloco *Registrador de Controle* fará a interceptação dos sinais da lógica de controle preexistente que estão à caminho da memória. Nessa interceptação verificará se houve a inserção, no módulo *DDR*, dos dados necessários para o cálculo da *Distância* L^2: a janela de busca e o vetor *NxNi*. Caso seja verificada a inserção dos dados pretendidos, o *Registrador de Controle* ativa o bloco de *Controle de Dados* e este, por sua vez, fará o controle da informação da memória que será enviada para efetuar o cálculo no bloco *Distância L2*. Concluído o cálculo desejado, o *Controle de Dados* envia os valores calculados para a memória do kit para que sejam resgatados pelo software através da lógica de controle preexistente, finalizando o processo desempenhado pelo hardware pretendido.

Para que o *Registrador de Controle* tenha conhecimento de um dado pronto para ser processado em memória, antes de tudo ele necessita verificar o valor do código da variável compartilhada com o software.

No hardware, o espaço determinado para a variável compartilhada foi o endereço zero da memória *DDR*. Portanto, quando o software alterar o valor da variável compartilhada, este valor será inserido no endereço zero da memória do kit. Após inserir o dado a ser processado na memória do hardware, o software irá lançar no endereço zero o código (0x00000000 00000001) para que o bloco *Distancia L2* inicie seu cálculo através do desvio do fluxo do sistema para o processamento interno do bloco *Controle de Dados*. Após o término do processamento, será lançado para o endereço zero o código de permissão de leitura da memória do kit (0x00000000 00000000).

Sendo assim, o trabalho desempenhado pelo *Registrador de Controle* é verificar o valor que está sendo lançado no espaço reservado para a variável compartilhada, responsável pelo desvio do fluxo do hardware para o processamento do dado em

memória, através da ativação e desativação de circuitos multiplexadores desenvolvidos, e pelo retorno ao fluxo original do sistema em hardware, através da finalização do processamento do bloco *Controle de Dados*. Tal lógica foi condensada no módulo *Registrador de Controle* representado na Figura 4.3.

Figura 4.3. Bloco *Registrador de Controle*.

Na Figura 4.3 observamos sinais de entrada azuis e vermelhos. Os sinais em azul representam sinais interceptados da arquitetura interna do projeto de referencia (ALTERA2, 2003). Os sinais vermelhos foram criados para o controle do processamento do bloco *Controle de Dados*.

O módulo *Registrador de Controle* desempenha a execução da máquina de estados que obedece à descrição da Figura 4.4. A figura obedece à descrição de uma *Máquina de Mealy* onde a entrada *S* refere-se ao estado de entrada e a saída *SP* indica o próximo estado da máquina.

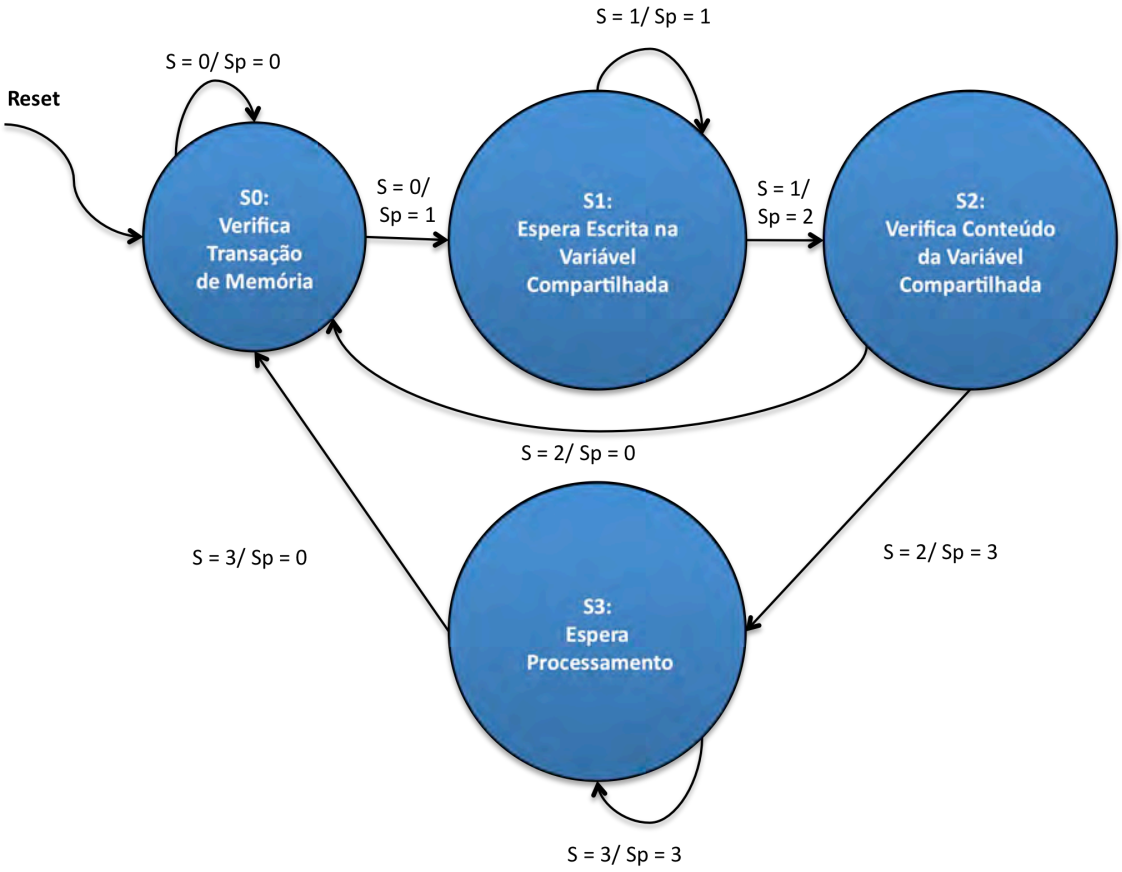


Figura 4.4. Máquina de estados do módulo *Registrador de Controle.*

Os estados da máquina descrita na Figura 4.4 são brevemente descritos abaixo:

- S0: Verifica Transação de Memória. Neste estado a máquina verifica a ativação dos sinais necessários para se haver um pedido de escrita no endereço zero da memória do kit, espaço da variável compartilhada. Caso haja a ativação dos sinais, vai para o estado *S1*. Caso contrário, continua na espera da ativação dos sinais;
- S1: Espera Escrita na Variável Compartilhada. Neste momento a máquina espera apenas que o valor da variável compartilhada seja escrito. Terminada a escrita, a máquina migra para o estado *S2*;
- S2: Verifica Conteúdo da Variável Compartilhada. Neste estado a máquina trata o valor presente na variável compartilhada. Caso haja o código de processamento dos dados da memória (código 0x00000000 00000001), ativa os circuitos multiplexadores que altera o fluxo do sistema para o bloco *Controle de Dados* e espera o fim do processamento do mesmo indo para o estado *S3*. Caso contrário, vai para o estado *S0* e espera uma nova transação de escrita no endereço zero;

- S3: Espera Processamento. Neste momento a máquina aguarda a finalização do bloco *Controle de Dados*. Finalizado o processamento, a máquina desativa os circuitos multiplexadores e retorna para o estado inicial esperando uma nova escrita na variável compartilhada.

Para o sistema em hardware pretendido concluir suas atividades, é necessário o auxílio do bloco *Controle de Dados*, que efetua a transferência dos dados da memória *DDR* do kit para o bloco *Distância L2* efetuar o seu processamento. O módulo *Controle de Dados* é representado pelo diagrama de bloco da Figura 4.5.

Figura 4.5. Bloco *Controle de Dados*.

Na Figura 4.5 observamos sinais de entrada azuis e vermelhos. Os sinais em azul representam sinais interceptados da arquitetura interna do projeto de referencia (ALTERA2, 2003). Os sinais vermelhos foram criados para o controle de seu processamento por meio do bloco *Registrador de Controle*.

O módulo *Controle de Dados* desempenha a execução da máquina de estados que obedece à descrição da Figura 4.6. A figura obedece à descrição de uma *Máquina de Mealy* onde a entrada *S* refere-se ao estado de entrada e a saída *SP* indica o próximo estado da máquina.

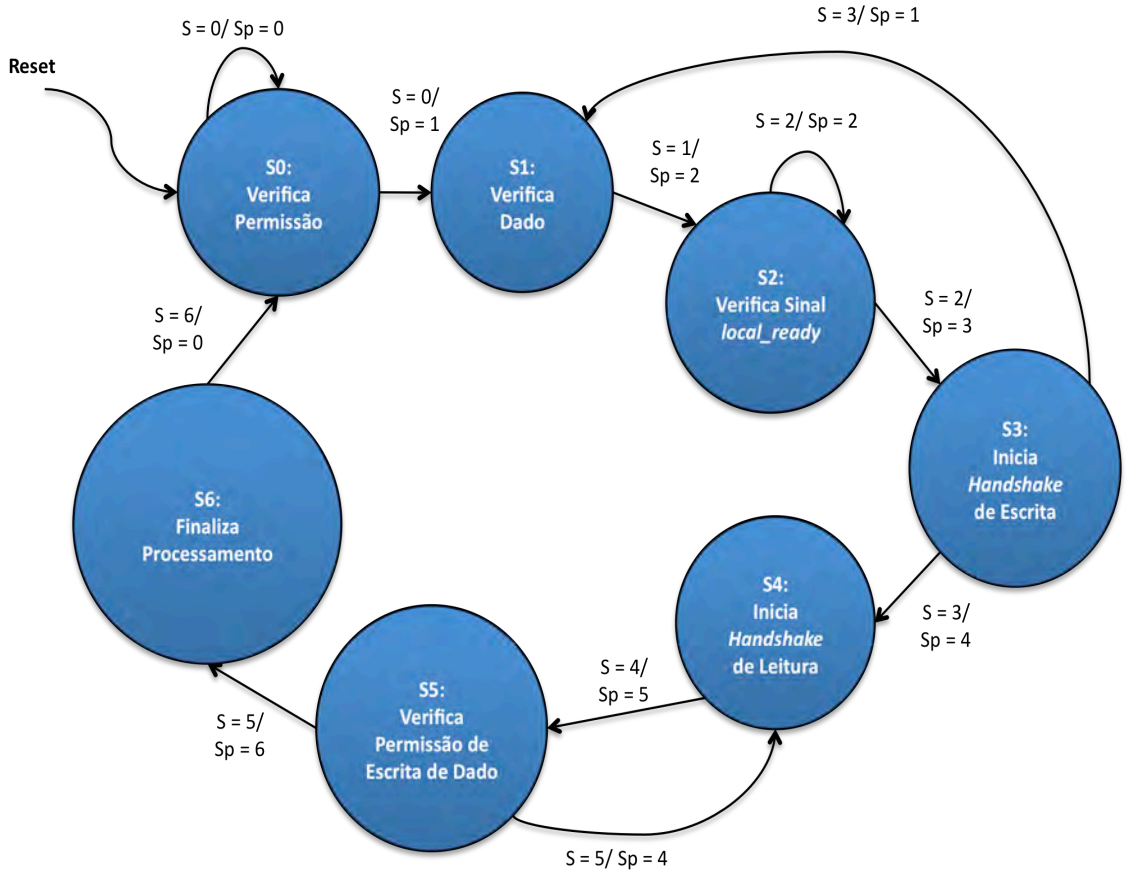


Figura 4.6. Máquina de estados do processamento do módulo *Controle de Dados.*

Os estados da máquina descrita na Figura 4.6 são brevemente descritos abaixo:

- S0: Verifica Permissão. Verifica se o *Registrador de Controle* liberou a permissão de processamento. Lembrando que o *Registrador de Controle* se baseia no valor da variável compartilhada. Caso haja permissão de processamento, a máquina vai para o próximo estado, se não continua no estado atual até haver permissão de processamento em hardware;
- S1: Verifica Dado. Neste estado a máquina inicia o procedimento de pegar os dados da memória e processá-los caso ainda haja dados a serem processados em memória;
- S2: Verifica Sinal *local_ready*. O estado verifica se ainda há dado a ser processado na memória, aguarda o sinal *local_ready* estar ativado e vai para o estado *S3.* Enquanto o sinal *local_ready* não é ativado, a máquina permanece neste estado a cada ciclo;
- S3: Inicia *Handshake* de Escrita. Neste estado a máquina inicia o processo de

handshake de escrita dos valores requisitados pelo bloco *Distancia L2* (vetor *NxNi* e janela de busca). Caso ainda haja dados a serem enviados para o bloco *Distancia L2*, a máquina retorna para o estado *S1*. Caso não haja mais dados para efetuar o *handshake* de escrita, a máquina vai para o estado *S4* para aguardar os resultados do bloco *Distância* L^2;

- S4: Inicia *Handshake* de Leitura. Aqui a máquina inicia o processo de *handshake* de leitura com o bloco *Distância L2* (resultados do cálculo da *Distância* L^2). Após cada rajada do processo de *handshake* efetuada, a máquina vai para o estado *S5* para iniciar o processo de escrita dos valores na memória;
- S5: Verifica Permissão de Escrita de Dado. Espera o sinal de permissão de escrita de dado estar ativado para iniciar a escrita no ciclo posterior. Tendo inserido os valores provenientes do cálculo da *Distância* L^2 na memória *DDR* do kit, a máquina se direciona novamente para o estado *S4* caso haja valores ainda por serem recebidos do bloco *Distancia L2*. Caso tenha finalmente cessado o fluxo de dados vindos do *Distancia L2*, vai ao estado *S6*;
- S6: Finaliza Processamento. Prepara os sinais da arquitetura para finalizar a atividade da máquina para o processamento atual, prepara o código de finalização do processamento em hardware (0x00000000 00000000) para escrevê-lo na variável e vai para o estado *S0* para aguardar o próximo processamento.

Antes da implementação do sistema proposto e para garantir que as modificações do hardware original na placa de desenvolvimento não alteraria o desempenho, principalmente com as mudanças de controles críticos, foi precedida uma simples implementação de inversão na imagem através de circuitos inversores.

No contexto de imagens digitais, o negativo de uma imagem nada mais é do que os valores invertidos dos pixels que a constituem. O cálculo do negativo se caracteriza através da Equação 4.1.

$$n(i) = L - v(i)$$

Equação 4.1. Cálculo do negativo em uma imagem digital.

Onde:

- ***n(i):*** Valor do negativo no pixel *i*;
- ***L:*** Valor máximo estabelecido pela resolução da imagem. Para imagens de 8 bits, semelhantes às que usamos, esse valor será de 255;
- ***v(i):*** Valor observado no pixel *i* da imagem original.

Sendo assim, ao enviar uma imagem para o kit e recuperar o valor de seu processamento, se obtém o negativo da imagem enviada. O teste foi efetuado com sucesso e o resultado é demonstrado na Figura 4.7, validando o circuito dos blocos customizados desenvolvidos.

(a) **(b)**

Figura 4.7 Resultado da inversão efetuada em hardware. (a) Imagem original, (b) Imagem resultante do processamento em hardware.

Com este exemplo finaliza-se o esclarecimento sobre as atividades do trabalho efetuadas sobre os módulos *Registrador de Controle* e *Controle de Dados*.

V. Resultados Obtidos

Os resultados apresentados nesta seção são baseados nas implementações desenvolvidas em hardware e software, até então, necessárias para se validar a idéia do sistema inicialmente pretendido.

Inicialmente, são exibidos e comentados os resultados de implementação do hardware. Logo em seguida, os resultados de implementação do software do sistema são exibidos, mostrando sua arquitetura final.

Por fim, são apresentados os resultados obtidos com as camadas de hardware e software atuando em conjunto para atingir a otimização pretendida. Uma comparação e discussão com trabalhos correlatos é efetuada, encerrando o capítulo.

Durante o desenvolvimento deste trabalho, os testes descritos foram realizados sobre uma máquina com arquitetura computacional *Intel Core i7 870 (8 MB Cache L2, 2.93 GHz), Windows 7 – 64 Bits*, utilizando as quatro imagens exibidas na Figura 5.1 (*Boats, Barbara, Lena e Peppers*), todas com ruído branco gaussiano com desvio padrão $\sigma = 5$, possuindo dimensões de 512x512 pixels e 8 bits de resolução.

(a) **(b)**

(c) **(d)**

Figura 5.1. Imagens utilizadas nos testes descritos. (a) *Boats*; (b) *Barbara*; (c) *Lena*; (d) *Peppers*.

5.1. Camada de Hardware

Para o desenvolvimento do hardware do sistema foi concluída a síntese da

descrição no dispositivo *EP1S60F1020C6* da *Altera*, tendo em vista que a placa de prototipação *PCI* oferece tal dispositivo para ser utilizado.

Para o dispositivo, um núcleo do filtro de ruídos desenvolvido ocupa 17% da lógica e menos de 1% da memória. Foi alcançada uma frequência máxima de operação de 81,59 *MHz* (período de relógio de aproximadamente 12,25 ns).

Para o mesmo dispositivo, quatro núcleos ocupam 84% da lógica e ainda utilizam menos de 1% da memória, sendo que nesse caso foi alcançada uma frequência máxima de operação de 73,96 *MHz* (período de relógio de aproximadamente 13,52 ns). Esses resultados são resumidos na Tabela 5.1.

Tabela 5.1. Resultados de Síntese.

	Lógica Utilizada	**Memória Utilizada**	**Freqüência (MHz)**
Um núcleo	17%	< 1%	81,59
Quatro núcleos	84%	< 1%	73,96

Com os resultados de síntese, optou-se pelo uso da abordagem do módulo que utiliza quatro núcleos visando ao máximo a aceleração do sistema.

Sendo assim, o módulo desenvolvido (*prot1*) é um protótipo do hardware para testes de desempenho do módulo que efetua o cálculo da *Distância* L^2. O mesmo é constituído de:

- Uma unidade de *Controle de Dados* para o gerenciamento dos dados que entram e saem no módulo;
- Quatro blocos *Distância L2* atuando em paralelo.

Os quatro blocos atuando em paralelo permitem o cálculo da *Distância* L^2 para quatro pixels diferentes da imagem. Com isso, há uma aceleração quatro vezes maior que o sistema executando apenas um único bloco. A Figura 5.2 demonstra em diagrama de blocos as conexões entre os blocos que constituem a arquitetura do sistema *prot1*.

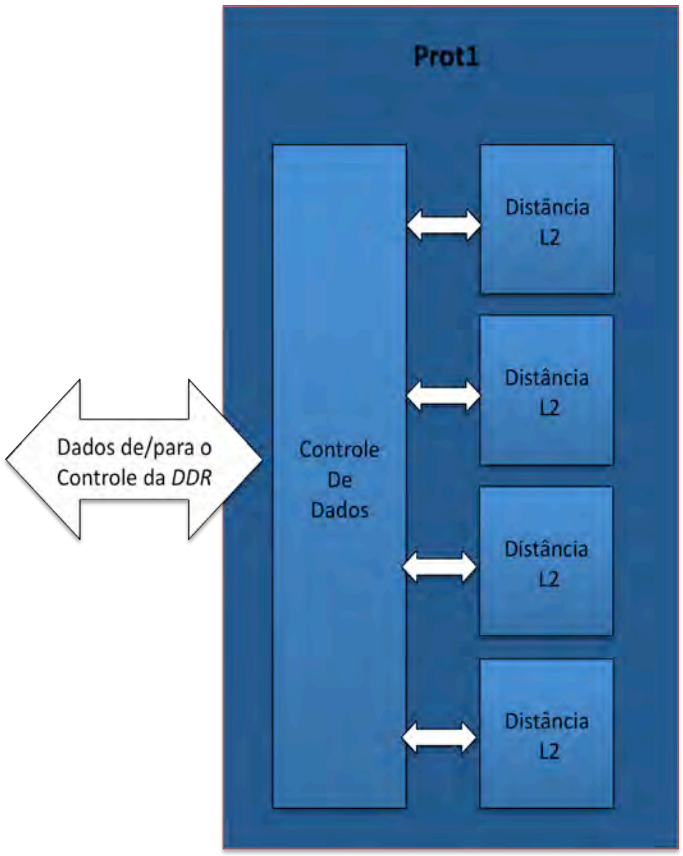


Figura 5.2. Arquitetura interna do bloco *prot1*.

Nos testes efetuados sobre o protótipo, o hardware efetua o cálculo da *Distância* L^2 para uma janela de busca em 68022 ciclos do sinal de *clock*. A quantidade de ciclos de *clock* para a conclusão do cálculo é fixa devido à abordagem empregada no desenvolvimento em *pipeline* das máquinas de estados de cada bloco interno que compõe o cálculo da *Distância* L^2.

De posse da quantidade de ciclos de *clock* necessários para finalizar o processamento paralelo das janelas de busca, é possível calcular o tempo de conclusão do processamento da *Distância* L^2 para várias quantidades de janelas de busca. Pode-se também estimar o tempo necessário que o hardware leva para calcular o desempenho do processamento para uma imagem completa e não apenas para uma janela de busca, dada uma certa freqüência de sinal de *clock*. Desse modo, obteve-se a Equação 5.1 descrita abaixo:

$$t = \frac{68022 \times R}{n \times f}$$

Equação 5.1. Estimativa do tempo de conclusão do cálculo da *Distância L^2* para uma imagem completa no hardware desenvolvido.

Onde:

- ***t:*** Tempo final para a conclusão do cálculo para uma imagem;
- ***R:*** Resolução da imagem desejada em pixels;
- ***f:*** Freqüência do sinal de *clock* aplicado ao protótipo do hardware;
- ***n:*** Constante relacionada à quantidade de núcleos da *Distância L2* que operam em paralelo. Neste trabalho serão quatro núcleos.

Utilizando a Equação 5.1 é possível estimar o tempo de execução necessário para executar o cálculo da *Distância L^2* utilizando o hardware desenvolvido.

Dessa forma, equiparando a freqüência de operação do hardware com a freqüência da arquitetura computacional utilizada nos testes, obteve-se estimativas de tempo de execução do cálculo da *Distância L^2*, sendo possível a comparação com o mesmo cálculo efetuado na abordagem de (BUADES, COLL e MOREL, 2004). As Figuras 5.3, 5.4, 5.5 e 5.6 exibem os gráficos comparativos entre a execução em hardware e em software do cálculo da *Distância L^2* sobre às imagens-teste citadas anteriormente na Figura 5.1.

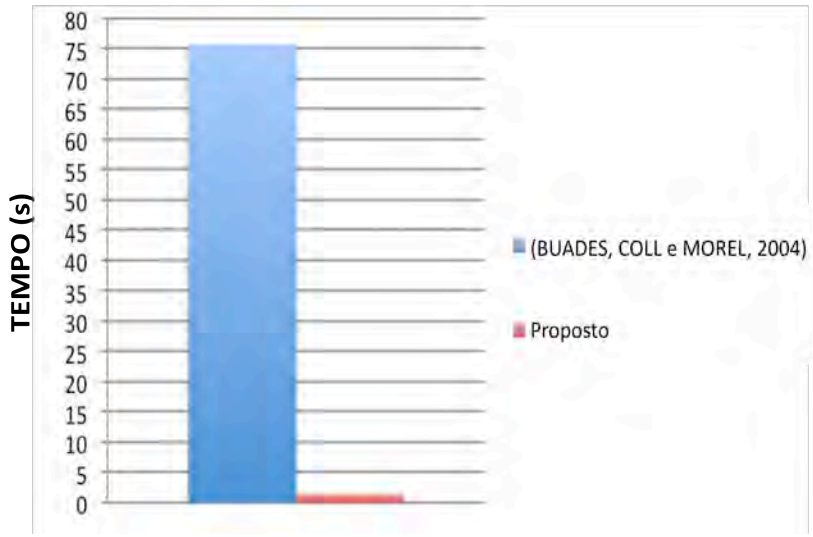


Figura 5.3. Desempenho do cálculo da *Distância* L^2 sobre a imagem *Boats.*

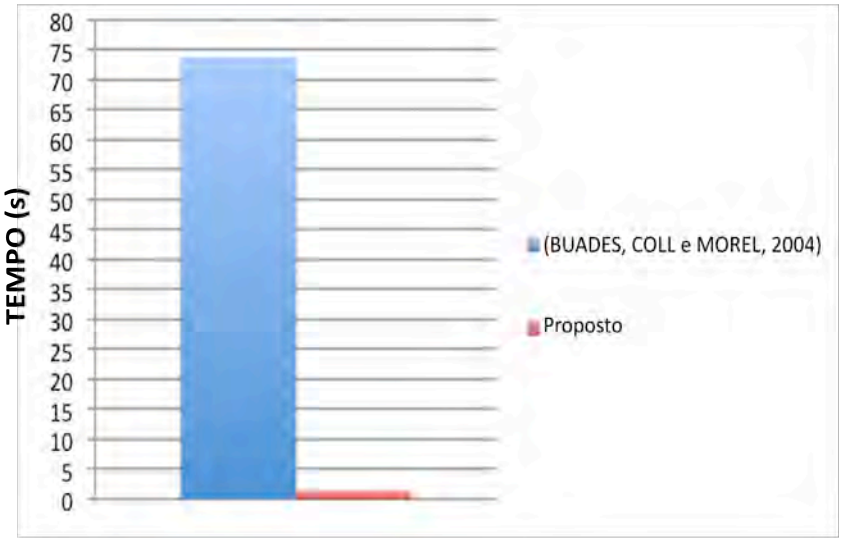


Figura 5.4. Desempenho do cálculo da *Distância* L^2 sobre a imagem *Barbara.*

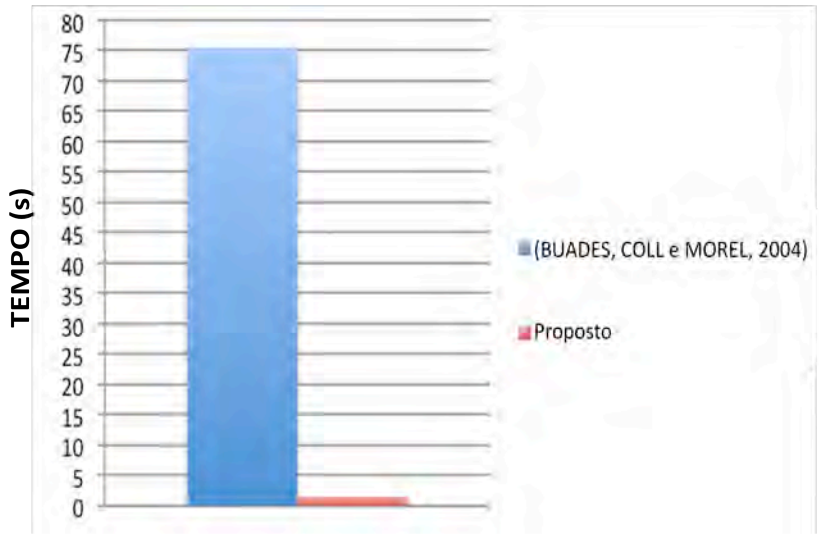


Figura 5.5. Desempenho do cálculo da *Distância L^2* sobre a imagem *Lena.*

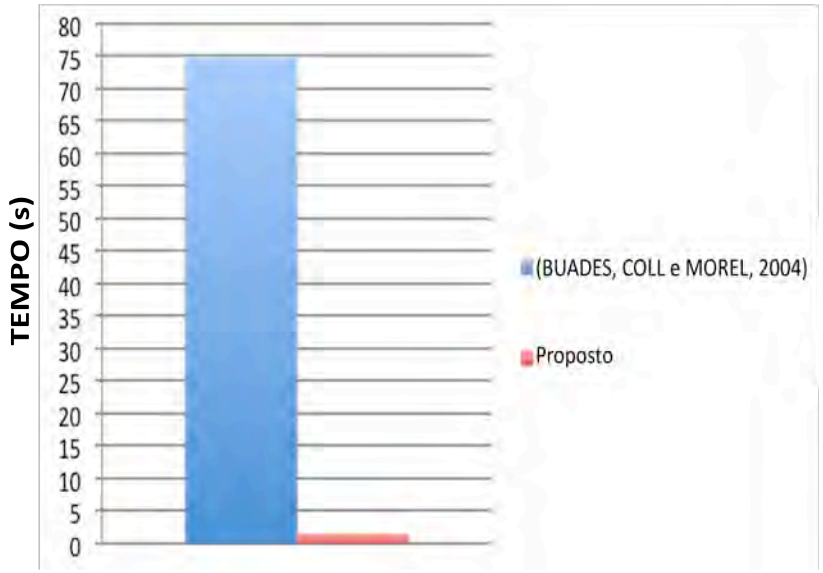


Figura 5.6. Desempenho do cálculo da *Distância L^2* sobre a imagem *Peppers.*

Nas imagens 5.3 à 5.6 observa-se que o trecho em software do cálculo da *Distância L^2* descrito em (BUADES, COLL e MOREL, 2004) é muito mais lento do que o cálculo do trabalho proposto, onde o hardware efetua o cálculo numa abordagem paralela atuando com *pipelines*.

5.2. Camada de Software

A parte concluída do software do sistema final corresponde à implementação do

código selecionado do algoritmo *NLM* – sem resolver o cálculo da *Distância* L^2 – integrada ao *driver* de comunicação com a interface *PCI*, como exibe o esquema da Figura 5.7.

O software do sistema é encarregado de preparar as janelas de busca da imagem e enviá-las ao hardware que efetua o cálculo da *Distância* L^2. Após o cálculo, o conteúdo é repassado novamente ao software para o mesmo continuar com os cálculos pendentes do *NLM* para o pixel em questão, como apresentado no diagrama de fluxo da Figura 5.8.

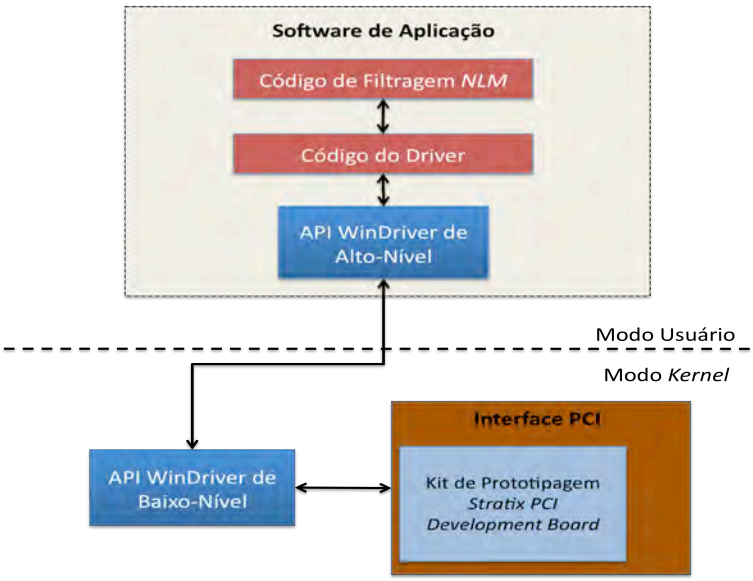


Figura 5.7. Arquitetura final da parte em software do sistema idealizado.

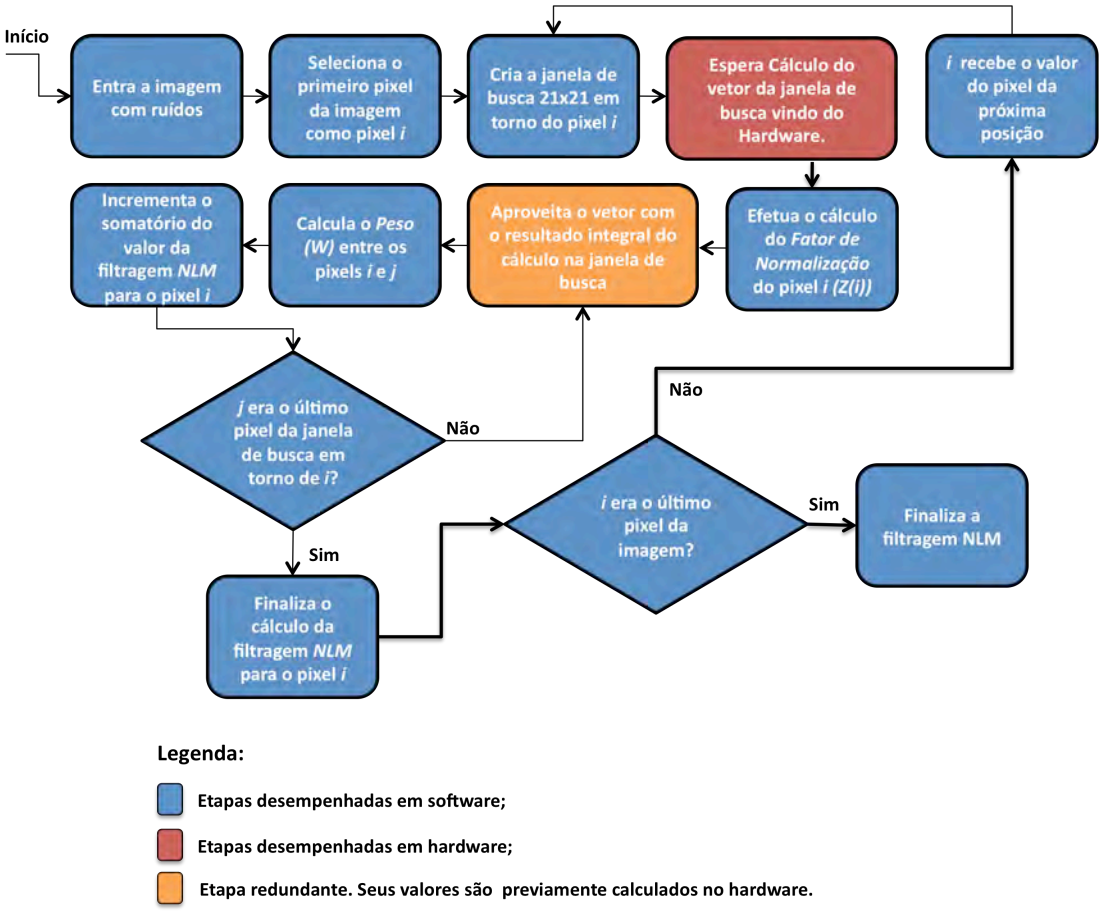


Figura 5.8. Fluxo final do software de filtragem *NLM*.

O *driver* realiza a comunicação entre a parte em software da abordagem pretendida da filtragem *NLM* e a interface *PCI* numa taxa de 66 MHz de freqüência por meio de rajadas de 64 bits.

Comparando o desempenho do software do sistema proposto com a abordagem descrita em (BUADES, COLL e MOREL, 2004) – a utilizada como referência para o desenvolvimento deste trabalho –, tem-se os gráficos comparativos das Figuras 5.9, 5.10, 5.11 e 5.12, referentes às imagens-teste citadas anteriormente na Figura 5.1. Nas figuras, o teste efetuado foi orientado apenas para as etapas desempenhadas em software, descritas na Figura 5.8, logo não computa o tempo para o cálculo da *Distância* L^2, este foi tratado na seção anterior.

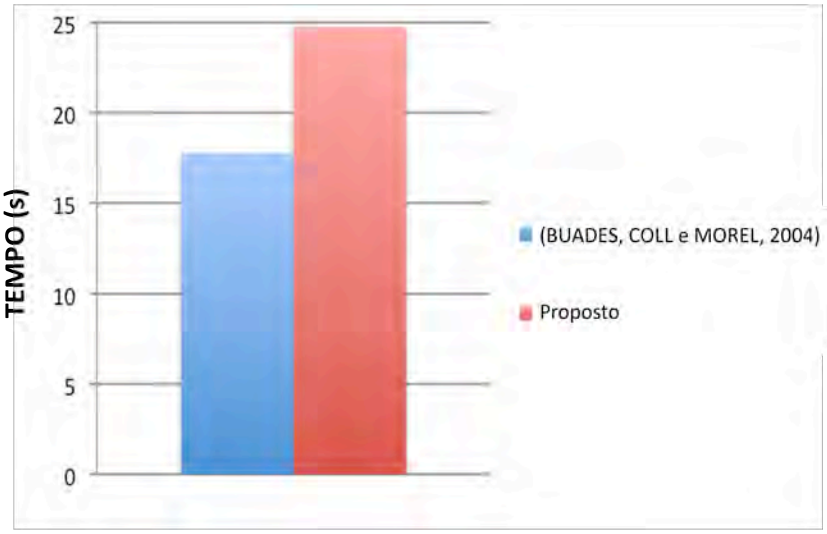


Figura 5.9. Desempenho do fluxo da parte do software durante a filtragem da imagem *Boats.*

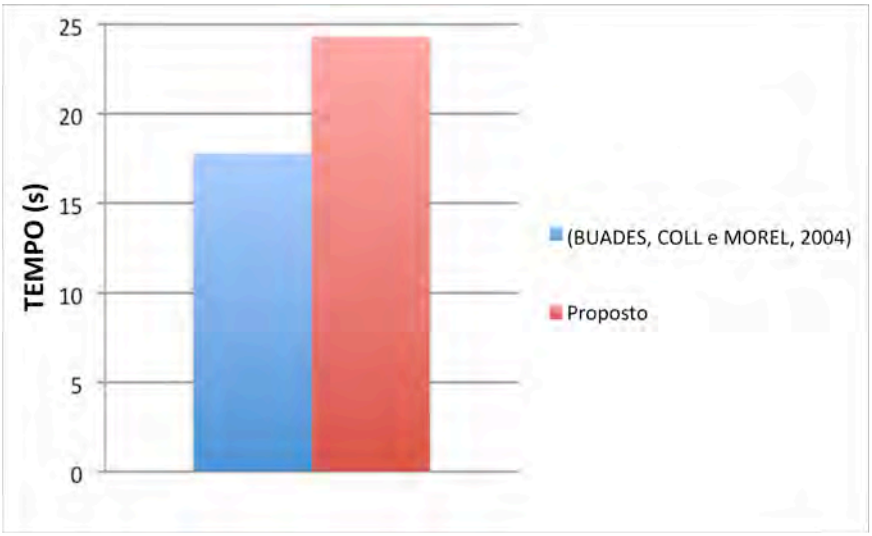


Figura 5.10. Desempenho do fluxo da parte do software durante a filtragem da imagem *Barbara.*

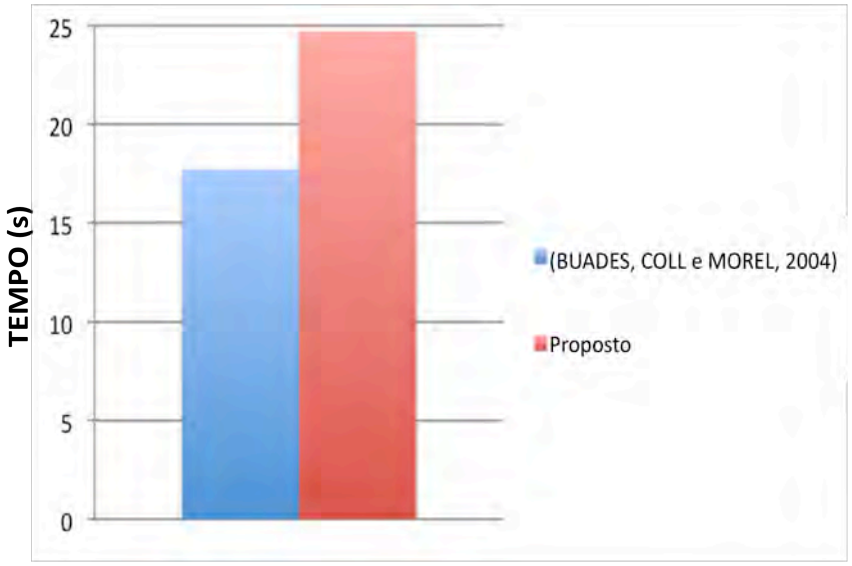


Figura 5.11. Desempenho do fluxo da parte do software durante a filtragem da imagem *Lena.*

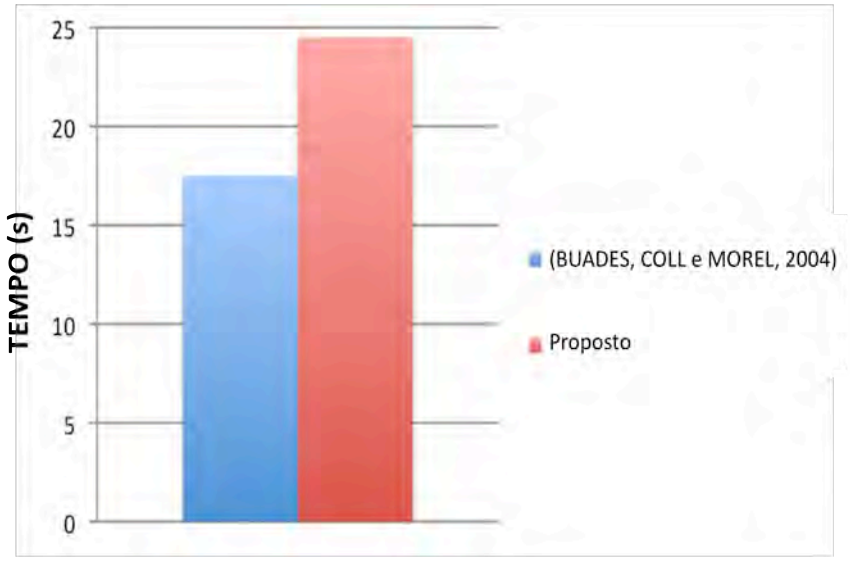


Figura 5.12. Desempenho do fluxo da parte do software durante a filtragem da imagem *Peppers.*

Nas imagens 5.9 à 5.12 observa-se que o trecho em software do trabalho proposto é mais lento que o mesmo trecho da abordagem de (BUADES, COLL e MOREL, 2004). Isso ocorre devido às requisições de acesso à *PCI* desempenhadas pelo *driver* do sistema proposto.

5.3. Hardware e Software: Resultados Integrados

De posse dos resultados do cálculo da *Distância* L^2, desempenhado em hardware, e dos cálculos desempenhados pelo software do sistema proposto, é possível verificar o tempo de conclusão da filtragem *NLM* desempenhada. As Figuras 5.13, 5.14, 5.15 e 5.16 exibem os gráficos comparativos entre a execução da filtragem *NLM* desempenhada pelo sistema proposto e a abordagem descrita em (BUADES, COLL e MOREL, 2004).

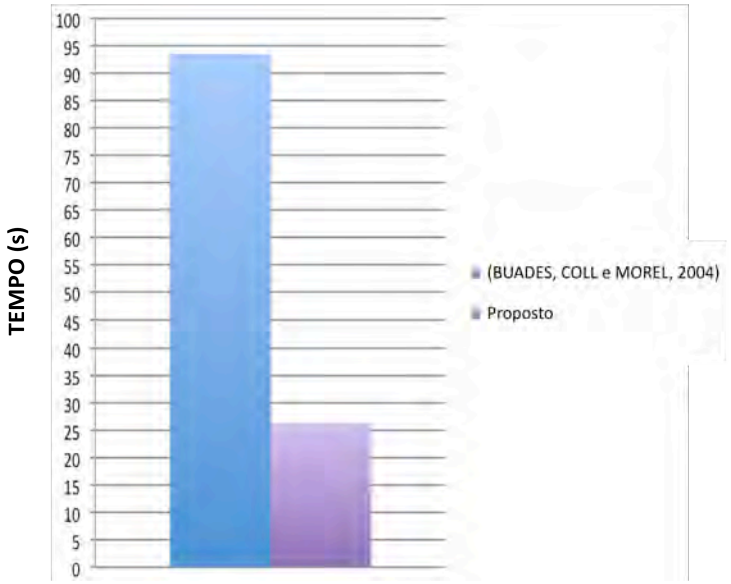


Figura 5.13. Desempenho do cálculo do *NLM* sobre a imagem *Boats.*

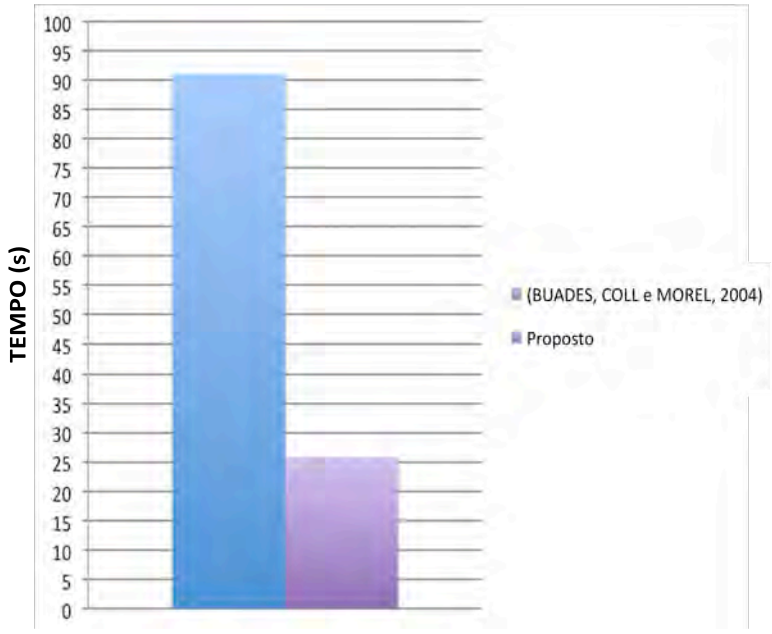


Figura 5.14. Desempenho do cálculo do *NLM* sobre a imagem *Barbara.*

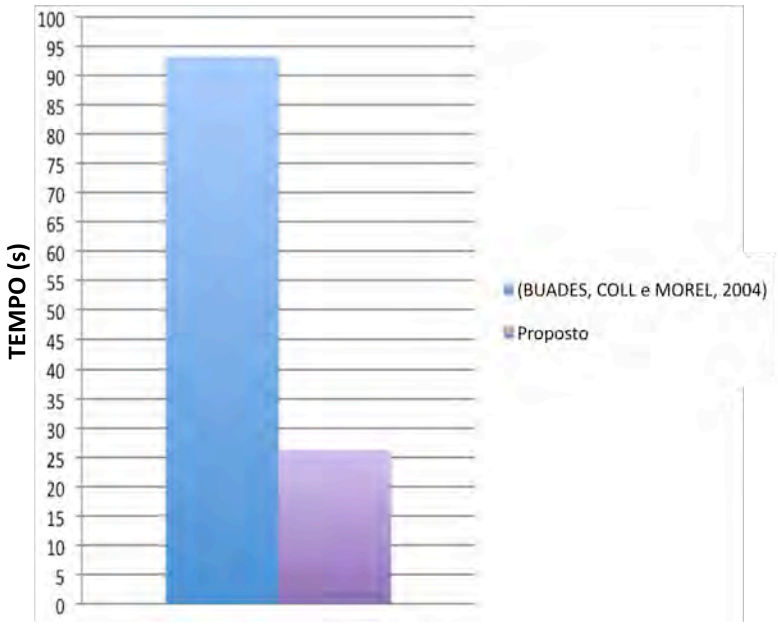


Figura 5.15. Desempenho do cálculo do *NLM* sobre a imagem *Lena.*

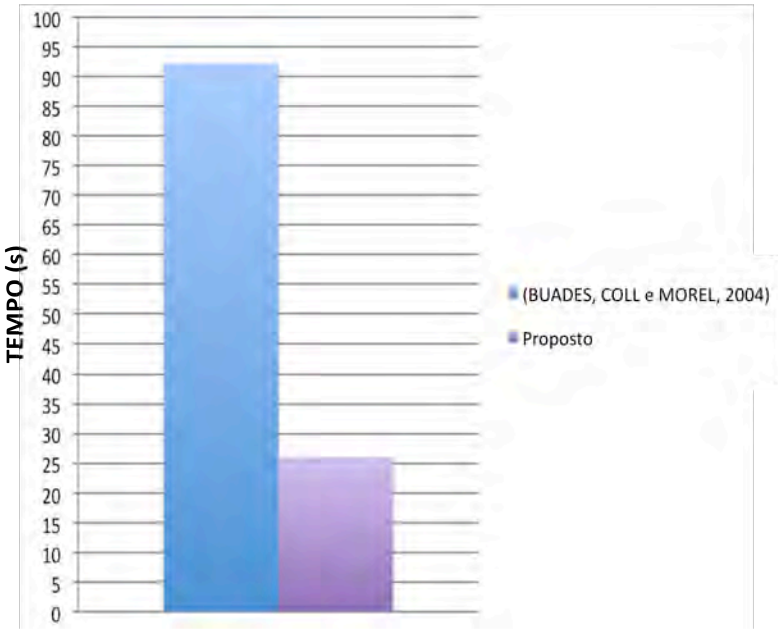


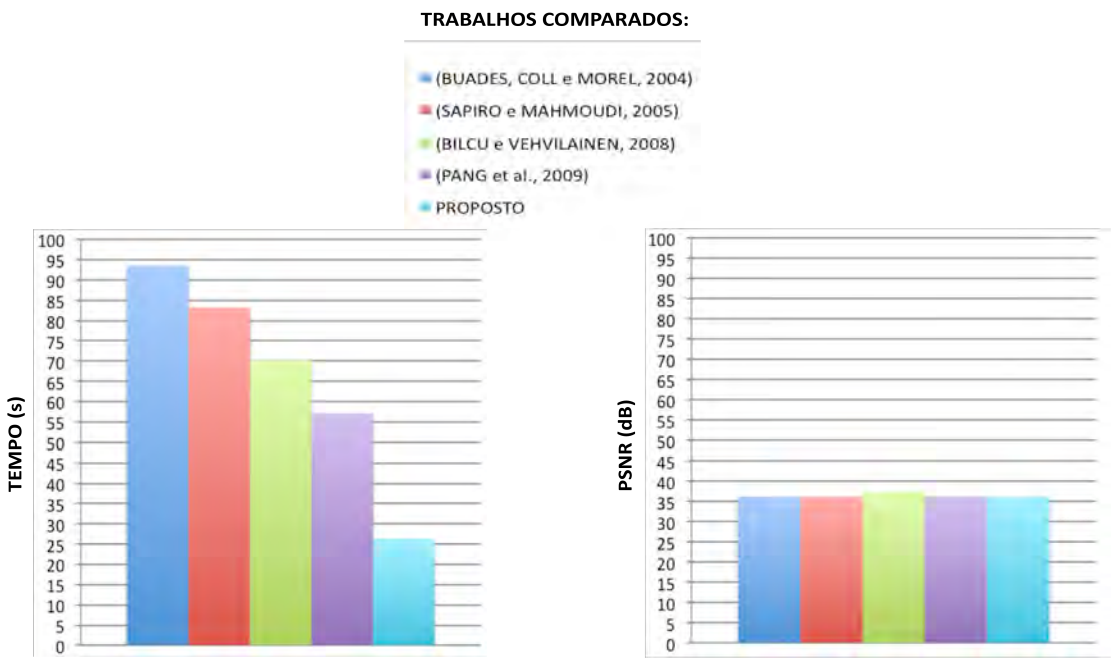
Figura 5.16. Desempenho do cálculo do *NLM* sobre a imagem *Peppers.*

Nas Figuras 5.13 à 5.16 observa-se que o processamento da filtragem *NLM* por meio da abordagem de (BUADES, COLL e MOREL, 2004) é mais lento do que o cálculo do trabalho proposto, onde há o processamento misto, em hardware e software.

5.4. Trabalhos Relacionados e Discussão Comparativa

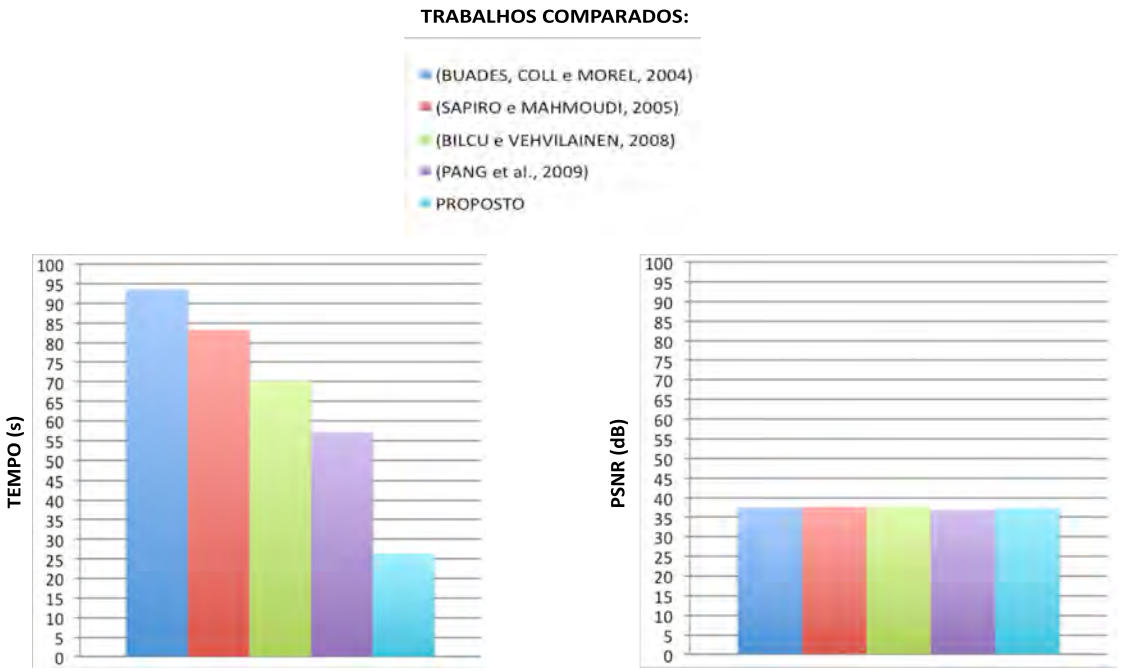
Além de (BUADES, COLL e MOREL, 2004), mais três trabalhos, já validados na literatura científica, foram selecionados para se comparar com o trabalho proposto. Os trabalhos selecionados são: (SAPIRO e MAHMOUDI, 2005), (BILCU e VEHVILAINEN, 2008), e (PANG et al., 2009). Estes possuem uma breve descrição da abordagem de otimização utilizada na seção 2.1.2.4 (*Otimizações do NLM*) desta dissertação.

Para comparar a eficiência dos resultados obtidos nos testes efetuados, mediu-se o tempo despendido para efetuar a execução do algoritmo *NLM* em cada abordagem e o índice *PSNR* de cada imagem filtrada, para servir de parâmetro de comparação de qualidade entre a imagem filtrada resultante e a original sem ruídos. As Figuras 5.17, 5.18, 5.19 e 5.20 exibem os gráficos comparativos desenvolvidos. A parte *(a)* de cada figura refere-se à comparação de tempo, enquanto que a parte *(b)* refere-se ao índice *PSNR* de cada método utilizado.



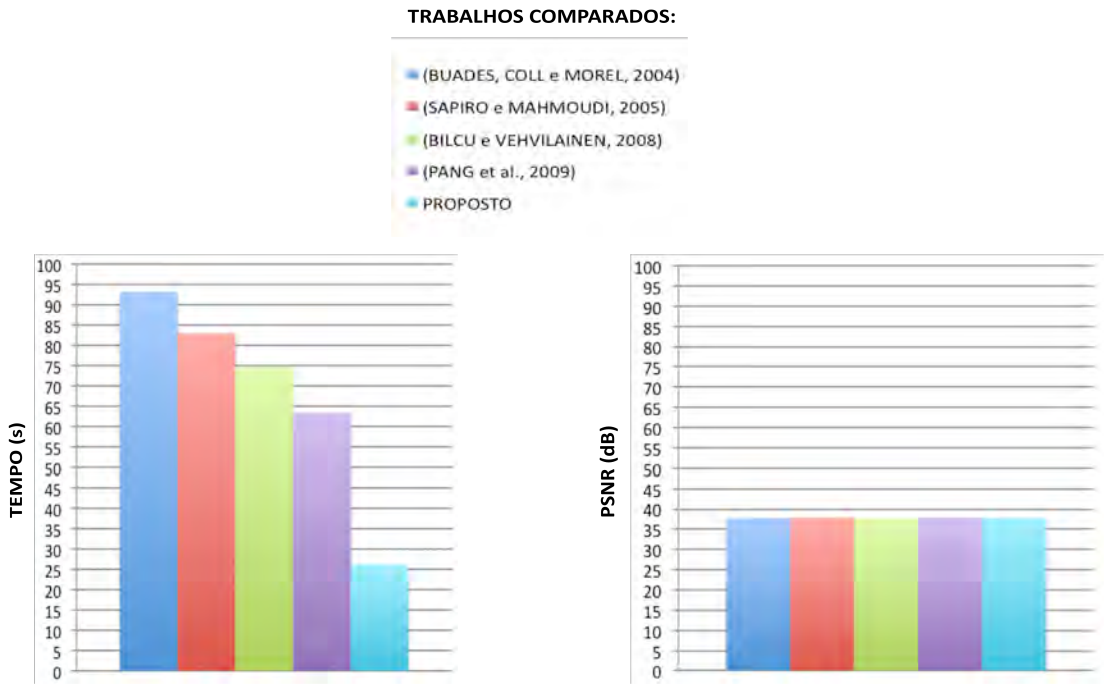
(a) **(b)**

Figura 5.17. Comparativos de tempo e índice *PSNR* para a imagem *Boats*.



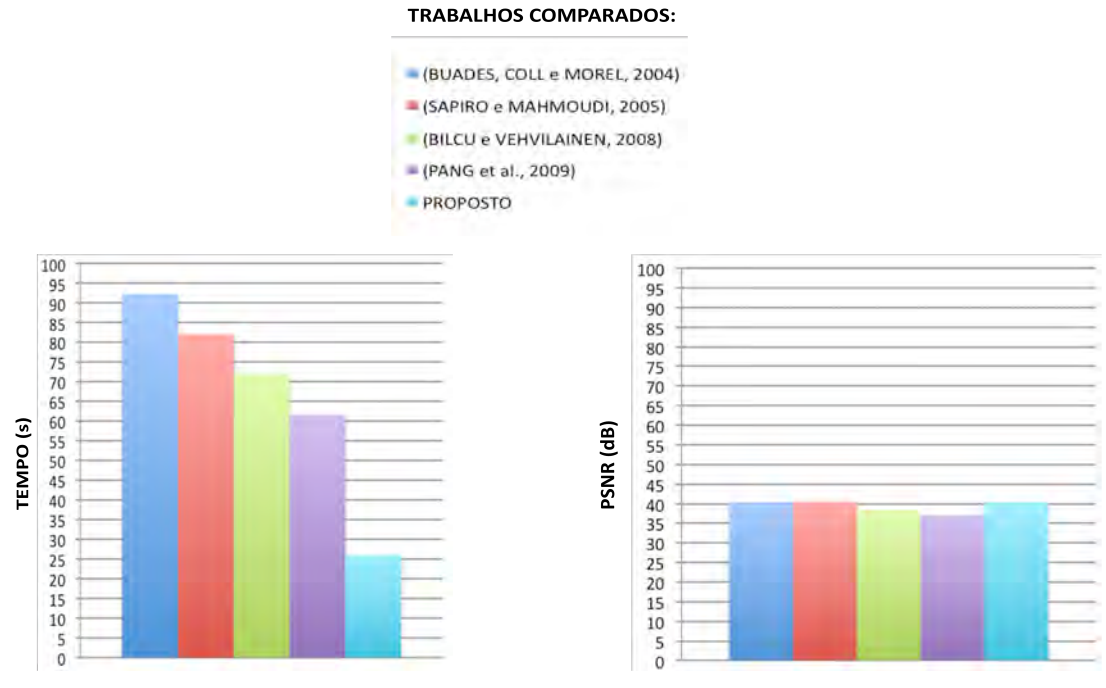
(a) **(b)**

Figura 5.18. Comparativos de tempo e índice *PSNR* para a imagem *Barbara*.



(a) **(b)**

Figura 5.19. Comparativos de tempo e índice *PSNR* para a imagem *Lena*.



(a) **(b)**

Figura 5.20. Comparativos de tempo e índice *PSNR* para a imagem *Peppers*.

Nas Figuras 5.17 à 5.20 é possível observar nos gráficos (*a*) a vantagem temporal da utilização da abordagem pretendida do sistema proposto. É visível a grande diferença entre esta abordagem e as outras, atingindo tempo de processamento inferior à metade da execução do segundo melhor tempo.

Observa-se também que o uso do método proposto nesta dissertação mantém praticamente a mesma qualidade visual dos outros métodos em questão – medida do índice *PSNR* nos gráficos (*b*) –, validando assim, a idéia do sistema pretendido.

A Tabela 5.2. resume o comparativo de tempo médio de filtragem das abordagens selecionadas.

Tabela 5.2. Tempo médio de filtragem dos trabalhos comparados.

Trabalho	**Abordagem Utilizada**	**Tempo Médio de Filtragem *NLM***
(BUADES, COLL e MOREL, 2004)	Janelas de busca.	92,51 s
(SAPIRO e MAHMOUDI, 2005)	Cálculos para pré-seleção de pixels.	82,33 s
(BILCU e VEHVILAINEN, 2008)	Interseção de intervalos de confiança.	70,36 s
(PANG et al., 2009)	Cálculos para pré-seleção de intervalos de confiança.	58,93 s
PROPOSTO	Sistema com hardware dedicado, conectado à interface *PCI*, para o cálculo da *Distância L2*.	26,09 s

VI. Considerações Finais

Neste trabalho, procurou-se acelerar a execução do algoritmo *NLM* por meio de uma abordagem que viesse a unir partes do algoritmo realizadas em software e partes realizadas em hardware paralelo, conectado à interface *PCI* do computador.

A abordagem de referência utilizada para a otimização do *NLM* obedece àquela já consagrada descrita em (BUADES, COLL e MOREL, 2004), que utiliza a metodologia de janelas de busca para o cálculo do grau de semelhança entre os pixels da imagem.

O trabalho resultante consiste numa abordagem em que há uma ação conjunta do software – responsável por selecionar os dados necessários da imagem de entrada – e do hardware – responsável por efetuar o cálculo dedicado, paralelizado através de *pipelines* superescalares, da *Distância Euclidiana Ponderada Quadrática*, retornando os valores resultantes ao software, para o mesmo concluir as etapas finais do algoritmo.

Os resultados obtidos nos testes, utilizando apenas quatro núcleos da *Distância L2*, exibem um ganho de tempo até 3.5 maior que as abordagens de otimização desenvolvidas apenas em software comparadas, mantendo também a métrica do índice *PSNR* estabilizada.

A abordagem utilizada pode ser explorada com o intuito de se otimizar ainda mais o desempenho e melhorar a qualidade visual da filtragem. Otimizações podem ser efetuadas tanto na implementação do hardware quanto no software do sistema.

6.1. Perspectiva de Trabalhos Futuros

Apesar da conclusão do sistema descrito no texto dessa dissertação, há a possibilidade de aprimorá-lo em suas partes de hardware e software, abrindo, dessa forma, possibilidades para trabalhos futuros.

Melhorias no hardware podem ser conseguidas através da implementação do sistema de uma forma que resulte em um menor consumo dos elementos lógicos, o que

possibilitaria a instanciação de mais núcleos do módulo responsável pelo cálculo da *Distância L2*, melhorando, assim, ainda mais a taxa de desempenho.

Outra idéia válida, para aceleração do hardware do sistema, é o desenvolvimento do *ASIC* do sistema que, segundo (HAUCK e DEHON, 2007), pode acelerar em até 25 vezes o tempo de processamento do sistema. Há também a possibilidade de síntese lógica do hardware utilizando tecnologias de dispositivos *FPGA* mais robustas e atuais. Por exemplo, testes preliminares de síntese do sistema foram efetuados sobre o dispositivo *5SGXEA7H3F35C3* da família *Stratix V* da *Altera*, resultando na utilização de apenas 3% da lógica do dispositivo, menos de 1% dos blocos de memória disponíveis e de uma freqüência de operação de 1 GHz.

Com relação ao software, novas abordagens de otimização do algoritmo *NLM* estão sendo pesquisadas com grande intensidade nos últimos anos, o que permite que o software do sistema possa ser adaptado a algumas dessas melhorias mais recentes para, assim, atingir resultados ainda melhores quando atuar em conjunto com o sistema em hardware. As abordagens utilizadas nas comparações podem ser utilizadas para a otimização do software do sistema, uma vez que o trabalho de referência utilizado (BUADES, COLL e MOREL, 2004) é anterior aos comparados (SAPIRO e MAHMOUDI, 2005), (BILCU e VEHVILAINEN, 2008) e (PANG et al., 2009).

Referências Bibliográficas

ALTERA. **Stratix PCI Development Board ver. 2.0**. ALTERA CORPORATION, 2003. 76p. (Data Sheet).

ALTERA2. **An 223: PCI-to-DDR SDRAM Reference Design ver. 1.0**. ALTERA CORPORATION, 2003. 34p. (Application Note 223).

ARM LIMITED. **AMBA AXI Protocol v1.0: Specification.** 2004. Disponível em: <http://www-micrel.deis.unibo.it/˜tocho/ferrara/Notes/Specifica.pdf>. Acesso em 24 mar. 2012.

ATHANAS, P.; SILVERMAN, H. F. **Processor Reconfiguration Through Instruction Set Metamorphosis: Compiler and Architecture**. 1993. IEEE Computer Society Press, v. 26, n. 3, p. 11-18.

AVANAKI, A. N.; DIYANAT, A.; SODAGARI, S. **Optimum parameter estimation for non-local means image de-noising using corner information.** 2008. 9th International Conference on Signal Processing, p. 861-863.

BARR, M. **Programming Embedded Systems in C and C++**. Sebastopol, CA - US: O'Reilly, 1999. 187p.

BERGER, A. **Embedded Systems Design**. Berkeley, CA - US: O'Reilly, 2002. 237p.

BILCU, R. C.; VEHVILAINEN, M. **Combined Non-Local averaging and intersection of confidence intervals for image de-noising.** 2008. 15th IEEE International Conference on Image Processing. p. 1736-1739.

BONATO, V.; MAZZOTI, B.; MARQUES, E. **Introdução: Técnicas de Co-projeto de Hardware e Software baseado em sistemas embarcados**, 2009.

BUADES, A.; COLL, B.; MOREL, J. M. **On image denoising methods**. 2004. 40p. Technical Note v. 5, CMLA.

DAUWE, A.; GOOSENS, B.; LUONG H. Q.; PHILIPS, W. **A fast non-local image denoising algorithm.** 2008. Image Processing: Algorithms and Systems VI (Proceedings Volume). Vol. 6812.

EFROS, A.; LEUNG, T. **Texture synthesis by non parametric sampling**. 1999. Seventh IEEE International Conference on Computer Vision. p. 1033-1038, v. 2.

GAMBARRA, L. L. **Otimização do algoritmo Non-local Means utilizando uma abordagem mista de hardware e software**. 2010. 54f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal da Paraíba, Campus I, João Pessoa.

GANSSLE, J. **The Art of Designing Embedded Systems**. 2nd Ed. San Diego, CA - US: Newnes Press Inc., 2008. 312p.

GONZALES, R. C.; WOODS, R. E. **Processamento de Imagens Digitais**. 1 Ed. São Paulo, SP - BR: Edgard Blücher, 2000. 509p.

GOOSSENS, B. et al. **A GPU-Accelerated Real-Time NLMeans Algorithm for Denoising Color Video Sequences**. 2010. Lecture Notes in Computer Science**,** v. 6475/2010, p. 46-57.

HEATH, S. **Embedded Systems Design**. 2nd Ed. Burlington, MA: Oxford Newnes, 2003. 451p.

HAUCK, S.; DEHON, A. **Reconfigurable Computing: The Theory and practice of FPGA-Based computation.** Burlington, MA: Morgan Kaufmann, 2007. 880p.

JUNGO. **WinDriverTM PCI/ISA/CardBus User's Manual ver. 10.2.1**. Jungo Ltd. 2010. 315p. (User's Manual).

OLUKOTUN, K. et al. **A software-hardware cosynthesis approach to digital system simulation**. 1994. IEEE Micro, v. 14, p. 48-58.

MAXFIELD, C. M. **The Design Warrior's Guide to FPGA: Devices, Tools and Flows.** Burlington, MA: Elsevier, 2004. 560p.

LAI, R.; DOU, X. **Improved Non-local Means Filtering Algorithm for Image Denoising.** 2010. 3rd International Congresse on Image and Signal Processing, v. 2, p. 720-722.

PANG, C. et al. **A fast NL-Means method in image denoising based on the similarity of spatially sampled pixels.** 2009. IEEE International Workshop on Multimedia Signal Processing. p. 1-4.

ROSSI, D. L. **Projeto de um controlador PID para controle de ganho de uma câmera com sensor CMOS utilizando computação reconfigurável.** 2012. 89f. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Universidade de São Paulo, São Paulo.

SAPIRO, G.; MAHMOUDI, M. **Fast image and video denoising via nonlocal means of similar neighborhoods**. 2005. IEEE Signal Processing Letters, v. 12, n. 12, p. 839-842.

SHAHAM, N. **Métodos para aceleração do "non-local means" algoritmo de redução de ruído.** 2007. 88f. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

SILVA, D. D. C. **Desenvolvimento de um IP-Core de pré-processamento digital de voz para aplicação em sistemas embutidos.** 2006**.** 108f**.** Dissertação (Mestrado em Informática) - Universidade Federal de Campina Grande, Campina Grande - PB.

SOUZA, A. R. C. **Desenvolvimento e implementação em FPGA de um sistema portátil para aquisição e compressão sem perdas de Eletrocardiogramas.** 2008. 180f. Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, João Pessoa - PB.

TENNENHOUSE, D. **Proactive Computing**. 2000. Communications of the ACM, v. 43, n. 5, p. 43-50.

TANENBAUM, A. S. **Organização Estruturada de Computadores.** 5 Ed. São Paulo: Prentice Hall, 2006. 464 p.

VAHID, F. **Sistemas Digitais: Projeto, Otimização e HDLs.** Porto Alegre: Artmed, 2008. 560 p.

VILLASENOR, J.; SMITH, W. H. M. **Configurable Computing**. 1997. Scientific America.

APÊNDICE A. Hardware Desenvolvido

Nesta seção serão exibidos os blocos componentes do hardware desenvolvido (*prot1*). Os módulos foram desenvolvidos na linguagem de descrição de hardware *VHDL* no ambiente *Quartus II*.

A.1. Módulo *Distância L2*

O módulo *Distância L2 é composto de* sete blocos: *vetorNxNi, janelaDeBusca, subtração, quadrado, soma 1, pondera e soma 2*. Neste ponto é apresentado um exemplo de descrição em *VHDL* (bloco *vetorNxNi*) e em seguida os blocos reproduzidos pelo *Quartus.* As Figuras A.1 e A.2 exibem os blocos internos do módulo *Distância L2* e seu módulo alto-nível.

Na imagem A.1 os blocos *vetorNxNi, janelaDeBusca, subtração, quadrado, soma 1, pondera e soma 2* são, respectivamente, representados pelos blocos marcados pelas letras *A, B, C, D, E, F* e *G*.

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity vetorNxNi is
	port (
		-- Canais AXI:
		clk_ddr : in std_logic;
		rstn    : in std_logic;
		--Canal de Endereço de Escrita:
		AWID    : in std_logic_vector(2 downto 0);  -- até 7.
		AWVALID : in std_logic;
		AWREADY : out std_logic;
		--Canal de Escrita de Dados:
		WID    : in std_logic_vector(2 downto 0);  -- até 7.
		WDATA  : in std_logic_vector(63 downto 0);
		WVALID : in std_logic;
		WREADY : out std_logic;
		--Canal de Resposta de Escrita:
```

```
            BID    : out std_logic_vector(2 downto 0);  -- até 7.
            BVALID : out std_logic;
            BREADY : in std_logic;
            --Canal de Endereço de Leitura:
            ARID    : in std_logic_vector(8 downto 0); -- Até 441
            ARVALID : in std_logic;
            ARREADY : out std_logic;
            --Canal de Leitura de Dados:
            RID     : out std_logic_vector(8 downto 0); -- Até 441.
            RVALID   : out std_logic;
            RREADY   : in std_logic;
            RDATA_00 : out std_logic_vector(7 downto 0);
            RDATA_01 : out std_logic_vector(7 downto 0);
            RDATA_02 : out std_logic_vector(7 downto 0);
            RDATA_03 : out std_logic_vector(7 downto 0);
            RDATA_04 : out std_logic_vector(7 downto 0);
            RDATA_05 : out std_logic_vector(7 downto 0);
            RDATA_06 : out std_logic_vector(7 downto 0);
            RDATA_07 : out std_logic_vector(7 downto 0);
            RDATA_08 : out std_logic_vector(7 downto 0);
            RDATA_09 : out std_logic_vector(7 downto 0);
            RDATA_10 : out std_logic_vector(7 downto 0);
            RDATA_11 : out std_logic_vector(7 downto 0);
            RDATA_12 : out std_logic_vector(7 downto 0);
            RDATA_13 : out std_logic_vector(7 downto 0);
            RDATA_14 : out std_logic_vector(7 downto 0);
            RDATA_15 : out std_logic_vector(7 downto 0);
            RDATA_16 : out std_logic_vector(7 downto 0);
            RDATA_17 : out std_logic_vector(7 downto 0);
            RDATA_18 : out std_logic_vector(7 downto 0);
            RDATA_19 : out std_logic_vector(7 downto 0);
            RDATA_20 : out std_logic_vector(7 downto 0);
            RDATA_21 : out std_logic_vector(7 downto 0);
            RDATA_22 : out std_logic_vector(7 downto 0);
            RDATA_23 : out std_logic_vector(7 downto 0);
            RDATA_24 : out std_logic_vector(7 downto 0);
            RDATA_25 : out std_logic_vector(7 downto 0);
            RDATA_26 : out std_logic_vector(7 downto 0);
            RDATA_27 : out std_logic_vector(7 downto 0);
            RDATA_28 : out std_logic_vector(7 downto 0);
            RDATA_29 : out std_logic_vector(7 downto 0);
            RDATA_30 : out std_logic_vector(7 downto 0);
            RDATA_31 : out std_logic_vector(7 downto 0);
            RDATA_32 : out std_logic_vector(7 downto 0);
            RDATA_33 : out std_logic_vector(7 downto 0);
            RDATA_34 : out std_logic_vector(7 downto 0);
            RDATA_35 : out std_logic_vector(7 downto 0);
            RDATA_36 : out std_logic_vector(7 downto 0);
            RDATA_37 : out std_logic_vector(7 downto 0);
            RDATA_38 : out std_logic_vector(7 downto 0);
            RDATA_39 : out std_logic_vector(7 downto 0);
            RDATA_40 : out std_logic_vector(7 downto 0);
            RDATA_41 : out std_logic_vector(7 downto 0);
            RDATA_42 : out std_logic_vector(7 downto 0);
            RDATA_43 : out std_logic_vector(7 downto 0);
            RDATA_44 : out std_logic_vector(7 downto 0);
            RDATA_45 : out std_logic_vector(7 downto 0);
            RDATA_46 : out std_logic_vector(7 downto 0);
            RDATA_47 : out std_logic_vector(7 downto 0);
            RDATA_48 : out std_logic_vector(7 downto 0)
      );
end vetorNxNi;

architecture processamento of vetorNxNi is
--Máquina de Estados:
constant ESPERA_END_ESCRITA                    : std_logic_vector(3 downto 0) := "0000";
```

```
constant ESPERA_DADO_ESCRITA                  : std_logic_vector(3 downto 0) := "0001";
constant ENVIA_RESP_ESCRITA                   : std_logic_vector(3 downto 0) := "0010";
constant ESPERA_RECEBIMENTO_RESP_ESCRITA : std_logic_vector(3 downto
0) := "0011";
constant ESPERA_END_LEITURA                   : std_logic_vector(3 downto 0) := "0100";
constant ENVIA_DADO_LEITURA                   : std_logic_vector(3 downto 0) := "0101";
constant ESPERA_RECEBIMENTO_DADO_LEITURA : std_logic_vector(3 downto
0) := "0110";
constant PREPARA_LEITURA                      : std_logic_vector(3 downto 0) := "0111";
constant REINICIA                             : std_logic_vector(3 downto 0) := "1000";
-- sinais internos da arquitetura do bloco:
signal estado     : std_logic_vector(3 downto 0);
signal idEscrita  : std_logic_vector(2 downto 0); -- até 7 escritas.
signal idLeitura  : std_logic_vector(8 downto 0); -- até 441 leituras.
signal dado               : std_logic_vector(63 downto 0);

begin
      process (clk_ddr, rstn)
   begin
            --Zera os sinais ao se resetar:
            if (rstn = '0') then
                  --Canal de Endereço de Escrita:
                  AWREADY <= '1';
                  --Canal de Escrita de Dados:
                  WREADY <= '0';
                  --Canal de Resposta de Escrita:
                  BID(2 downto 0) <= (others => '0');
                  BVALID <= '0';
                  --Canal de Endereço de Leitura:
                  ARREADY <= '0';
                  --Canal de Leitura de Dados:
                  RID(2 downto 0) <= (others => '0');
                  RVALID <= '0';
                  RDATA_00(7 downto 0) <= (others => '0');
                  RDATA_01(7 downto 0) <= (others => '0');
                  RDATA_02(7 downto 0) <= (others => '0');
                  RDATA_03(7 downto 0) <= (others => '0');
                  RDATA_04(7 downto 0) <= (others => '0');
                  RDATA_05(7 downto 0) <= (others => '0');
                  RDATA_06(7 downto 0) <= (others => '0');
                  RDATA_07(7 downto 0) <= (others => '0');
                  RDATA_08(7 downto 0) <= (others => '0');
                  RDATA_09(7 downto 0) <= (others => '0');
                  RDATA_10(7 downto 0) <= (others => '0');
                  RDATA_11(7 downto 0) <= (others => '0');
                  RDATA_12(7 downto 0) <= (others => '0');
                  RDATA_13(7 downto 0) <= (others => '0');
                  RDATA_14(7 downto 0) <= (others => '0');
                  RDATA_15(7 downto 0) <= (others => '0');
                  RDATA_16(7 downto 0) <= (others => '0');
                  RDATA_17(7 downto 0) <= (others => '0');
                  RDATA_18(7 downto 0) <= (others => '0');
                  RDATA_19(7 downto 0) <= (others => '0');
                  RDATA_20(7 downto 0) <= (others => '0');
                  RDATA_21(7 downto 0) <= (others => '0');
                  RDATA_22(7 downto 0) <= (others => '0');
                  RDATA_23(7 downto 0) <= (others => '0');
                  RDATA_24(7 downto 0) <= (others => '0');
                  RDATA_25(7 downto 0) <= (others => '0');
                  RDATA_26(7 downto 0) <= (others => '0');
                  RDATA_27(7 downto 0) <= (others => '0');
                  RDATA_28(7 downto 0) <= (others => '0');
                  RDATA_29(7 downto 0) <= (others => '0');
                  RDATA_30(7 downto 0) <= (others => '0');
                  RDATA_31(7 downto 0) <= (others => '0');
                  RDATA_32(7 downto 0) <= (others => '0');
```

```
        RDATA_33(7 downto 0) <= (others => '0');
        RDATA_34(7 downto 0) <= (others => '0');
        RDATA_35(7 downto 0) <= (others => '0');
        RDATA_36(7 downto 0) <= (others => '0');
        RDATA_37(7 downto 0) <= (others => '0');
        RDATA_38(7 downto 0) <= (others => '0');
        RDATA_39(7 downto 0) <= (others => '0');
        RDATA_40(7 downto 0) <= (others => '0');
        RDATA_41(7 downto 0) <= (others => '0');
        RDATA_42(7 downto 0) <= (others => '0');
        RDATA_43(7 downto 0) <= (others => '0');
        RDATA_44(7 downto 0) <= (others => '0');
        RDATA_45(7 downto 0) <= (others => '0');
        RDATA_46(7 downto 0) <= (others => '0');
        RDATA_47(7 downto 0) <= (others => '0');
        RDATA_48(7 downto 0) <= (others => '0');
        --Flags do sistema:
        dado(63 downto 0)        <= (others => '0');
        idEscrita(2 downto 0) <= (others => '0');
        idLeitura(8 downto 0) <= (others => '0');
        estado <= ESPERA_END_ESCRITA;
    --Se for evento de clock, vai varrendo os estados da máquina:
    elsif (rising_edge(clk_ddr)) then
        if (estado = ESPERA_END_ESCRITA) then
            if (AWVALID = '1') then
                idEscrita <= AWID;
                AWREADY   <= '0';
                WREADY    <= '1';
                estado    <= ESPERA_DADO_ESCRITA;
            end if;
        elsif (estado = ESPERA_DADO_ESCRITA) then
            if (WVALID = '1' and WID = idEscrita) then
                dado <= WDATA;
                WREADY <= '0';
                estado <= ENVIA_RESP_ESCRITA;
            end if;
        elsif (estado = ENVIA_RESP_ESCRITA) then
            if (idEscrita = "000") then
                RDATA_00 <= dado(7 downto 0);
                RDATA_01 <= dado(15 downto 8);
                RDATA_02 <= dado(23 downto 16);
                RDATA_03 <= dado(31 downto 24);
                RDATA_04 <= dado(39 downto 32);
                RDATA_05 <= dado(47 downto 40);
                RDATA_06 <= dado(55 downto 48);
                RDATA_07 <= dado(63 downto 56);
            elsif (idEscrita = "001") then
                RDATA_08 <= dado(7 downto 0);
                RDATA_09 <= dado(15 downto 8);
                RDATA_10 <= dado(23 downto 16);
                RDATA_11 <= dado(31 downto 24);
                RDATA_12 <= dado(39 downto 32);
                RDATA_13 <= dado(47 downto 40);
                RDATA_14 <= dado(55 downto 48);
                RDATA_15 <= dado(63 downto 56);
            elsif (idEscrita = "010") then
                RDATA_16 <= dado(7 downto 0);
                RDATA_17 <= dado(15 downto 8);
                RDATA_18 <= dado(23 downto 16);
                RDATA_19 <= dado(31 downto 24);
                RDATA_20 <= dado(39 downto 32);
                RDATA_21 <= dado(47 downto 40);
                RDATA_22 <= dado(55 downto 48);
                RDATA_23 <= dado(63 downto 56);
            elsif (idEscrita = "011") then
                RDATA_24 <= dado(7 downto 0);
```

```
            RDATA_25 <= dado(15 downto 8);
            RDATA_26 <= dado(23 downto 16);
            RDATA_27 <= dado(31 downto 24);
            RDATA_28 <= dado(39 downto 32);
            RDATA_29 <= dado(47 downto 40);
            RDATA_30 <= dado(55 downto 48);
            RDATA_31 <= dado(63 downto 56);
        elsif (idEscrita = "100") then
            RDATA_32 <= dado(7 downto 0);
            RDATA_33 <= dado(15 downto 8);
            RDATA_34 <= dado(23 downto 16);
            RDATA_35 <= dado(31 downto 24);
            RDATA_36 <= dado(39 downto 32);
            RDATA_37 <= dado(47 downto 40);
            RDATA_38 <= dado(55 downto 48);
            RDATA_39 <= dado(63 downto 56);
        elsif (idEscrita = "101") then
            RDATA_40 <= dado(7 downto 0);
            RDATA_41 <= dado(15 downto 8);
            RDATA_42 <= dado(23 downto 16);
            RDATA_43 <= dado(31 downto 24);
            RDATA_44 <= dado(39 downto 32);
            RDATA_45 <= dado(47 downto 40);
            RDATA_46 <= dado(55 downto 48);
            RDATA_47 <= dado(63 downto 56);
        elsif (idEscrita = "110") then
            RDATA_48 <= dado(7 downto 0);
        end if;
        BID    <= idEscrita;
        BVALID <= '1';
        estado <= ESPERA_RECEBIMENTO_RESP_ESCRITA;
    elsif (estado = ESPERA_RECEBIMENTO_RESP_ESCRITA) then
        if (BREADY = '1') then
            BVALID <= '0';
            if (idEscrita < "110") then
                estado  <= ESPERA_END_ESCRITA;
                AWREADY <= '1';
            else
                ARREADY <= '1';
                estado  <= ESPERA_END_LEITURA;
            end if;
        end if;

    elsif (estado = ESPERA_END_LEITURA) then
        if (ARVALID = '1' and ARID = idLeitura) then
            ARREADY <= '0';
            estado  <= ENVIA_DADO_LEITURA;
        end if;
    elsif (estado = ENVIA_DADO_LEITURA) then
        RVALID <= '1';
        RID    <= idLeitura;
        estado <= ESPERA_RECEBIMENTO_DADO_LEITURA;
    elsif (estado = ESPERA_RECEBIMENTO_DADO_LEITURA) then
        if (RREADY = '1') then
            RVALID <= '0';
            if (idLeitura < "110111000") then
                ARREADY   <= '1';
                idLeitura <= idLeitura + '1';
                estado    <= ESPERA_END_LEITURA;
            else
                estado <= REINICIA;
            end if;
        end if;
    elsif (estado = REINICIA) then
        --Canal de Endereço de Escrita:
    AWREADY <= '1';
```

```
        --Canal de Escrita de Dados:
        WREADY <= '0';
        --Canal de Resposta de Escrita:
        BID(2 downto 0) <= (others => '0');
        BVALID <= '0';
        --Canal de Endereço de Leitura:
        ARREADY <= '0';
        --Canal de Leitura de Dados:
        RID(2 downto 0) <= (others => '0');
        RVALID <= '0';
        RDATA_00(7 downto 0) <= (others => '0');
        RDATA_01(7 downto 0) <= (others => '0');
        RDATA_02(7 downto 0) <= (others => '0');
        RDATA_03(7 downto 0) <= (others => '0');
        RDATA_04(7 downto 0) <= (others => '0');
        RDATA_05(7 downto 0) <= (others => '0');
        RDATA_06(7 downto 0) <= (others => '0');
        RDATA_07(7 downto 0) <= (others => '0');
        RDATA_08(7 downto 0) <= (others => '0');
        RDATA_09(7 downto 0) <= (others => '0');
        RDATA_10(7 downto 0) <= (others => '0');
        RDATA_11(7 downto 0) <= (others => '0');
        RDATA_12(7 downto 0) <= (others => '0');
        RDATA_13(7 downto 0) <= (others => '0');
        RDATA_14(7 downto 0) <= (others => '0');
        RDATA_15(7 downto 0) <= (others => '0');
        RDATA_16(7 downto 0) <= (others => '0');
        RDATA_17(7 downto 0) <= (others => '0');
        RDATA_18(7 downto 0) <= (others => '0');
        RDATA_19(7 downto 0) <= (others => '0');
        RDATA_20(7 downto 0) <= (others => '0');
        RDATA_21(7 downto 0) <= (others => '0');
        RDATA_22(7 downto 0) <= (others => '0');
        RDATA_23(7 downto 0) <= (others => '0');
        RDATA_24(7 downto 0) <= (others => '0');
        RDATA_25(7 downto 0) <= (others => '0');
        RDATA_26(7 downto 0) <= (others => '0');
        RDATA_27(7 downto 0) <= (others => '0');
        RDATA_28(7 downto 0) <= (others => '0');
        RDATA_29(7 downto 0) <= (others => '0');
        RDATA_30(7 downto 0) <= (others => '0');
        RDATA_31(7 downto 0) <= (others => '0');
        RDATA_32(7 downto 0) <= (others => '0');
        RDATA_33(7 downto 0) <= (others => '0');
        RDATA_34(7 downto 0) <= (others => '0');
        RDATA_35(7 downto 0) <= (others => '0');
        RDATA_36(7 downto 0) <= (others => '0');
        RDATA_37(7 downto 0) <= (others => '0');
        RDATA_38(7 downto 0) <= (others => '0');
        RDATA_39(7 downto 0) <= (others => '0');
        RDATA_40(7 downto 0) <= (others => '0');
        RDATA_41(7 downto 0) <= (others => '0');
        RDATA_42(7 downto 0) <= (others => '0');
        RDATA_43(7 downto 0) <= (others => '0');
        RDATA_44(7 downto 0) <= (others => '0');
        RDATA_45(7 downto 0) <= (others => '0');
        RDATA_46(7 downto 0) <= (others => '0');
        RDATA_47(7 downto 0) <= (others => '0');
        RDATA_48(7 downto 0) <= (others => '0');
        --Flags do sistema:
        dado(63 downto 0)        <= (others => '0');
        idEscrita(2 downto 0) <= (others => '0');
        idLeitura(8 downto 0) <= (others => '0');
        estado <= ESPERA_END_ESCRITA;
        end if;
    end if;
```

```
    end process;
end processamento;
```

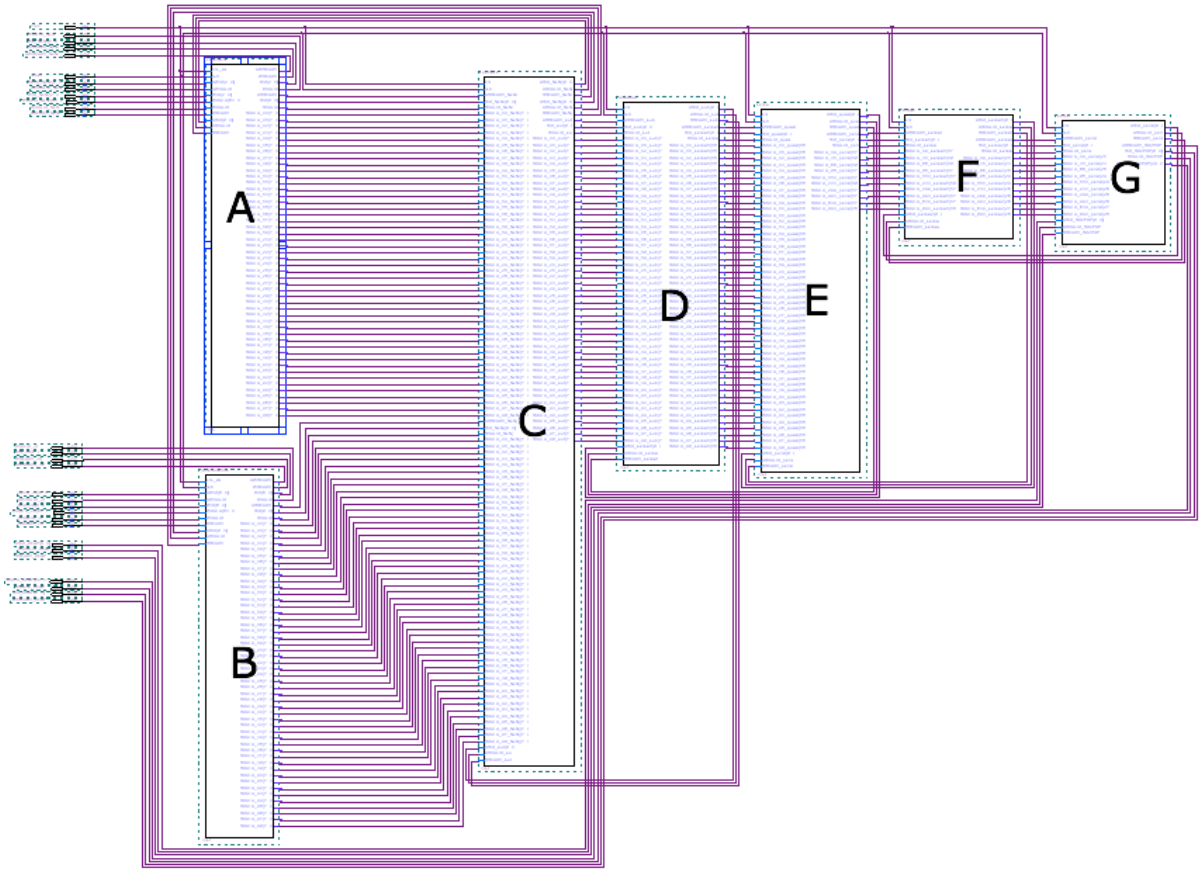


Figura A.1. Estrutura dos blocos internos do módulo *Distância L2*.

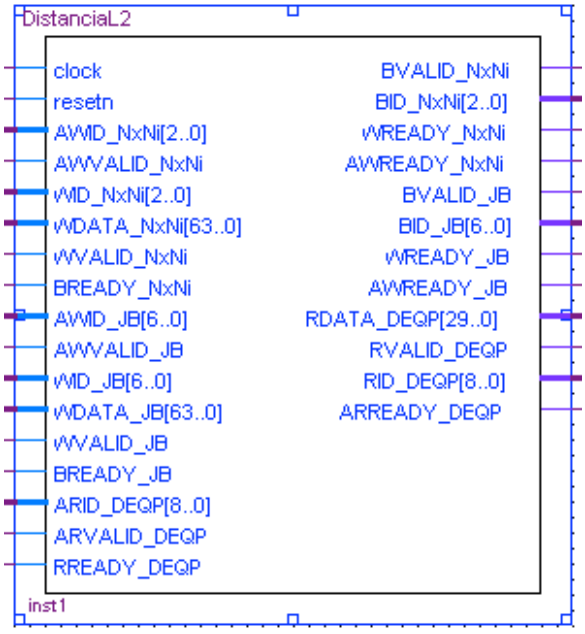


Figura A.2. Bloco alto-nível do módulo *Distância L2*.

A.2. Módulo de *Controle de Dados*

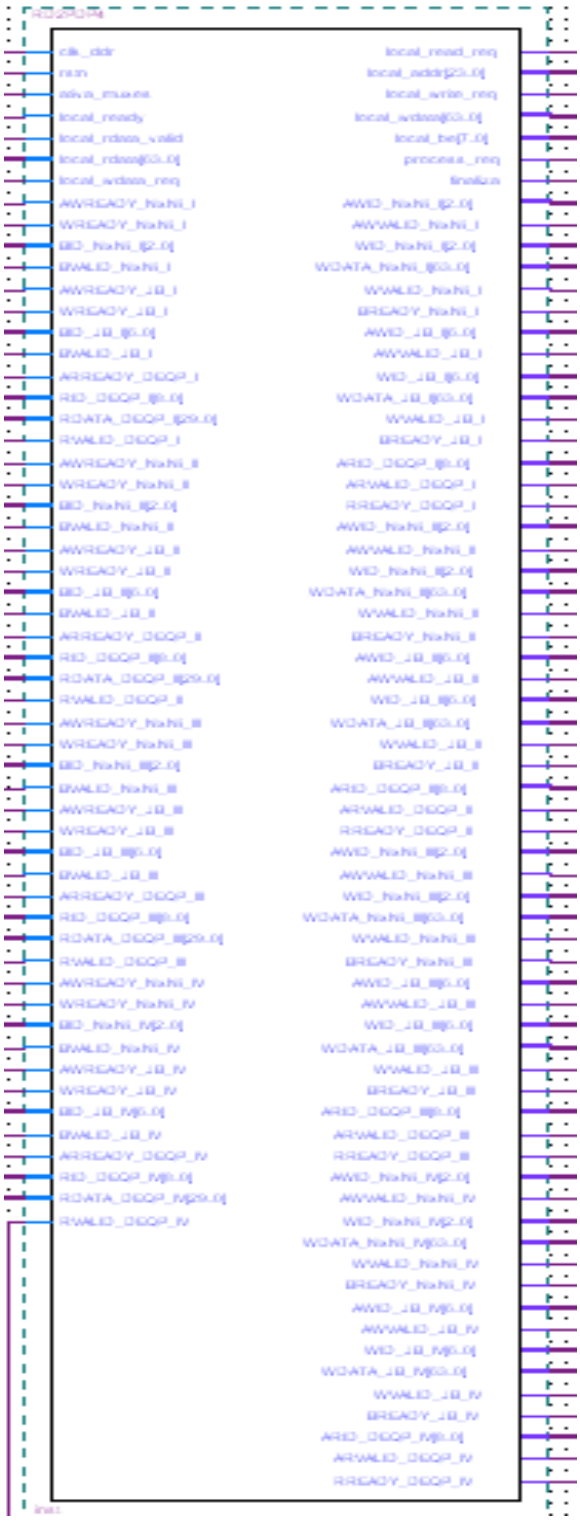


Figura A.3. Bloco do módulo *Controle de Dados*.

A.3. Módulo *prot1*

O módulo *prot1* é composto por um módulo de *Controle de Dados* e quatro módulos *Distância L2*. Na Figura A.4 o bloco de *Controle de Dados* é sinalizado com a letra *A*, enquanto que os módulos da *Distância L2* estão evidenciados pela letra *B.*

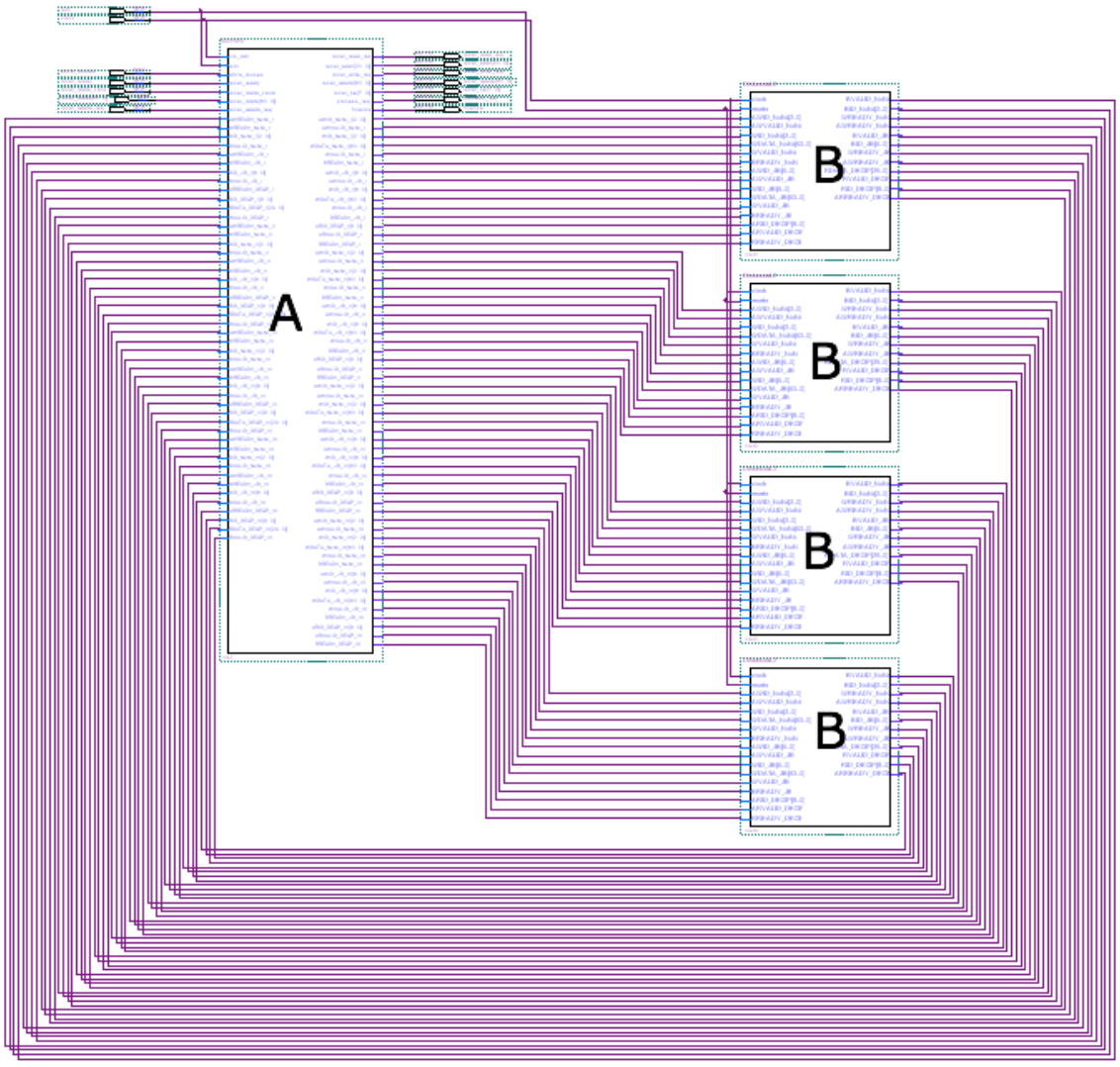


Figura A.4. Estrutura dos blocos internos do módulo *prot1*.