Universidade Federal de Pernambuco
Centro de Informática

Graduação em Engenharia de Computação

Detecção de Retinopatia Diabética Utilizando Uma Arquitetura de Hardware/Software

Gabriel de França Medeiros

Trabalho de Graduação

Recife
18 de dezembro de 2017

Universidade Federal de Pernambuco
Centro de Informática

Gabriel de França Medeiros

Detecção de Retinopatia Diabética Utilizando Uma Arquitetura de Hardware/Software

Trabalho apresentado ao Programa de Graduação em Engenharia de Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Adriano Augusto de Moraes Sarmento*

Recife
18 de dezembro de 2017

Dedico este trabalho a você que reservou parte do seu tempo para lê-lo.

Agradecimentos

Agradeço primeiramente aos meus pais, que sempre me apoiaram e se sacrificaram para me dar condições de chegar onde eu cheguei. Se um dia eu obtiver sucesso, eles terão 50% do mérito, mas se um dia eu fracassar, terei, sozinho, 100% da culpa.

Agradeço a todos os mestres, professores e orientadores, que tive durante a graduação. Sou grato por terem me guiado e por terem feito parte da minha graduação.

E por fim, e não menos importante, agradeço a todos os meus amigos. Agradeço por terem aturado minhas piadas ruins e por terem me ajudado a superar todos os desafios da graduação.

As pessoas fortes não derrubam as outras, elas ajudam-nas a se erguerem.
—GOKU (Dragon Ball Z)

Resumo

A retinopatia diabética é uma das principais causas de cegueira em pessoas portadoras de diabetes millitus se não for tratada corretamente. Este trabalho consiste em avaliar o algoritmo de verificação da severidade desta doença com base na imagem do fundo do olho que foi proposto por alunos de Iniciação Científica do Centro de Informática, da Universidade Federal de Pernambuco, verificar as partes mais custosas do algoritmo e implementar essas partes em HDL para ser executadas em FPGA. O algoritmo pode ser dividido em três partes principais: Detecção do disco óptico, detecção da fóvea e detecção dos exsudatos. Os experimentos iniciais foram divididos em 2 etapas. A primeira etapa foi medir o tempo médio de execução das 3 principais partes do algoritmo, citados anteriormente. Na segunda etapa, foi feito o cálculo do tempo médio de execução dos passos da parte do algoritmo com maior tempo médio de execução. A partir do resultado dos experimentos, concluiu-se que a etapa mais adequada para ser implementada neste trabalho era a dilatação utilizada na detecção do disco óptico. Após a implementação, foi realizada uma análise comparativa entre o sistema desenvolvido e a implementação original. O resultado desta análise mostra que existe melhora quando parte do algoritmo é acelerado em FPGA e o desempenho pode ser melhorado ainda mais se outras partes do algoritmo forem executadas diretamente em hardware.

Palavras-chave: retinopatia diabética, FPGA, RIFFA

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

1.1 Motivação

Retinopatia diabética é composto por um conjunto característico de lesões localizado na retina de indivíduos portadores de diabetes *mellitus* por muitos anos [Org17] e pode causar cegueira se não tratada propriamente. Esse tipo de lesão é chamada de edema macular diabético (EMD). A presença de exsudatos é um indicativo de que o paciente possui EMD e sua severidade depende da localização desses exsudatos. A localização dos exsudatos retinianos é de extrema importância na avaliação da severidade da doença. Quanto mais perto do centro da mácula (ponto conhecido como fóvea), mais grave é a doença. EMD pode ser classificado como suave, moderado ou severo [WSM10]. A Figura 1.1 mostra a retina de um paciente normal e a retina de um paciente com retinopatia diabética severa. O paciente diabético possui um grande número de exsudatos (manchas amarelas) próximo à mácula (parte mais escura da retina).

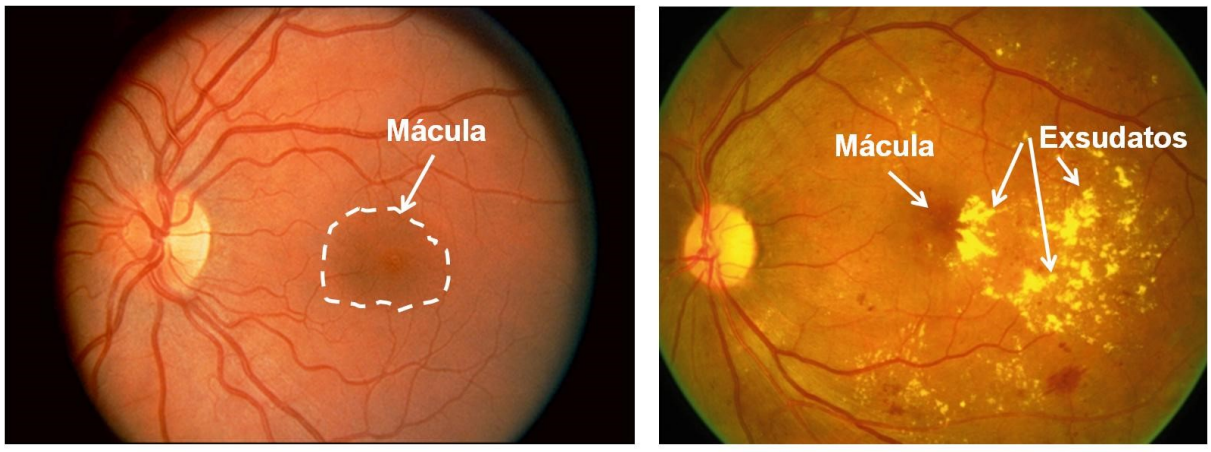


Figura 1.1 (a) Imagem do fundo de olho normal com mácula destacada (b) imagem de fundo de olho com retinopatia diabética, exsudatos são destacados.

A localização precisa dos exsudatos é essencial para o êxito do tratamento. É importante observar que o tipo de tratamento depende da gravidade do quadro, podendo variar de uma simples aplicação de laser ou até mesmo um tratamento cirúrgico das complicações.O método mais comumente utilizado para verificar a presença de exsudatos é a análise do fundo do olho usando um oftalmoscópio. Essa técnica depende da percepção visual do oftalmologista. Algumas técnicas foram sugeridas para aumentar a precisão, diminuir o tempo necessário para a

detecção de exsudatos e automatizar o processo, como os trabalhos de Sophorak [SUBW08] [SUB09] [SDU+10] e Welfer [WSM10].

Para avaliar a gravidade, o médico deve observar a distância entre os exsudatos e a fóvea. Uma distância inferior a $500\mu m$ indica uma retinopatia grave. O acompanhamento do paciente por parte do médico consiste, então, em verificar se a quantidade de exsudatos aumenta e se a distância entre eles e a fóvea se mantém estável. Caso a distância diminua, isto indica que a doença está mal controlada.

Os métodos atuais demandam o uso de equipamentos caros e de difícil transporte, o que impossibilita o diagnóstico e, consequentemente, o tratamento da retinopatia em habitantes de regiões de difícil acesso ou com rede elétrica de má qualidade. Além disso, demandam tempo para um especialista analisar as imagens e fazer dar um diagnóstico. Tempo é muito importante, principalmente nas redes públicas de saúde, onde a quantidade de pacientes é muito grande. Por esses motivos, um sistema embarcado que auxilia o diagnóstico de retinopatia pode causar um grande impacto positivo na luta contra esta doença.

Os métodos propostos atualmente focam na melhoria da acurácia em vez de custo computacional e tempo de execução. A alta complexidade desses métodos dificulta uma implementação embarcada, pois sistemas embarcados possuem limitações de processamento e memória. Um algoritmo desenvolvido como trabalho de iniciação científica por Gabriel Wanderley Albuquerque Silva e Thiago Soares de Melo, alunos da Universidade Federal de Pernambuco, tem como objetivo otimizar o processamento e o uso de memória na verificação da severidade de EMD.

1.2 Objetivo

O objetivo deste trabalho é analisar o algoritmo proposto pelos alunos Gabriel Silva e Thiago Melo e verificar quais passos do algoritmo teriam maior ganho de eficiência em tempo de execução e processamento se implementados em linguagem de descrição de hardware (HDL) e executados em FPGA.

É importante enfatizar que não está no escopo deste trabalho melhorar a acurácia do algoritmo. O algoritmo não será modificado, apenas parte do algoritmo será implementado utilizando uma tecnologia diferente.

Após a implementação do algoritmo em HDL, foram realizados experimentos com o objetivo de comparar a implementação puramente em software com a implementação utilizando uma abordagem hardware/software. Os experimentos irão considerar o tempo de resposta de todo o processo.

1.3 Estrutura

Com o objetivo de alcançar os objetivos apresentados, este trabalho possui a seguinte estrutura: O capítulo 2 aborda a fundamentação teórica necessária para o entendimento do trabalho realizado. Nele será explicado o algoritmo base do projeto e funcionamento do RIFFA, framework de integração utilizado no trabalho. O capítulo 3 apresenta o estado da arte, descrevendo técnicas utilizadas para diagnosticar a retinopatia e algoritmos de detecção de exsudatos. O capítulo

4 aborda os experimentos iniciais, que tiveram como objeto a seleção da etapa do algoritmo a ser implementada em linguagem de descrição de hardware e os módulos desenvolvidos, descrevendo seu funcionamento e como estão relacionados. No capítulo 5 é feita a descrição da avaliação de desempenho do sistema desenvolvido, apresentando a metodologia utilizada e os resultados obtidos. Por fim, o capítulo 6 apresenta a conclusão e os possíveis trabalhos futuros.

CAPÍTULO 2

Fundamentação Teórica

Este capítulo tem como objetivo apresentar conceitos necessários para o entendimento do trabalho. Na primeira seção, serão explicadas as operações morfológicas de dilatação, erosão, abertura e fechamento, usadas durante o algoritmo de verificação de severidade de EDM. Na segunda seção, será descrito o algoritmo no qual este trabalho se baseou, detalhando cada etapa e descrevendo as operações aplicadas sobre a imagem em cada passo do algoritmo. Na terceira seção será falado sobre o RIFFA, framework utilizado para implementar a comunicação entre o processador e o FPGA.

2.1 Operações Morfológicas

Operações morfológicas são operações aplicadas ao pixel definidas pela sua vizinhança. Essa vizinhança é definida por um conjunto bem definido chamado de elemento estruturante. Nesta seção serão abordados 4 operações morfológicas básicas: (1) dilatação, (2) erosão, (3) abertura e (4) fechamento [RR92].

As operações morfológicas percorrem toda a imagem e operam sobre cada pixel e sua vizinhança. Esta vizinhança é determinada por um elemento estruturante, que determina os pesos dos pixels na operação. O elemento estruturante possui valores que são multiplicados pelos valores dos pixels da imagem, os valores resultantes são utilizados na operação. A figura 2.1 mostra um exemplo do funcionamento do elemento estruturante.

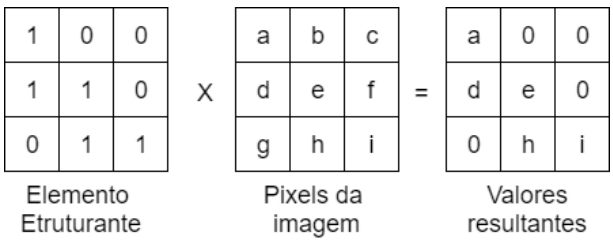


Figura 2.1 Exemplo de utilização de um elemento estruturante sobre um pixel (e) e sua vizinhança.

2.1.1 Dilatação

A dilatação consiste em uma operação morfológica onde o valor do pixel é substituído pelo valor máximo entre a vizinhança definida por um elemento estruturante. A operação de dilatação expande objetos da imagem, sendo comumente usada para preencher espaços entre elementos da imagem. A Figura 2.2 mostra o resultado da operação de dilatação sobre a imagem X utilizando o elemento estruturante quadrado 3x3 B. É perceptível a expansão dos objetos da imagem e o preenchimento do espaço entre eles.

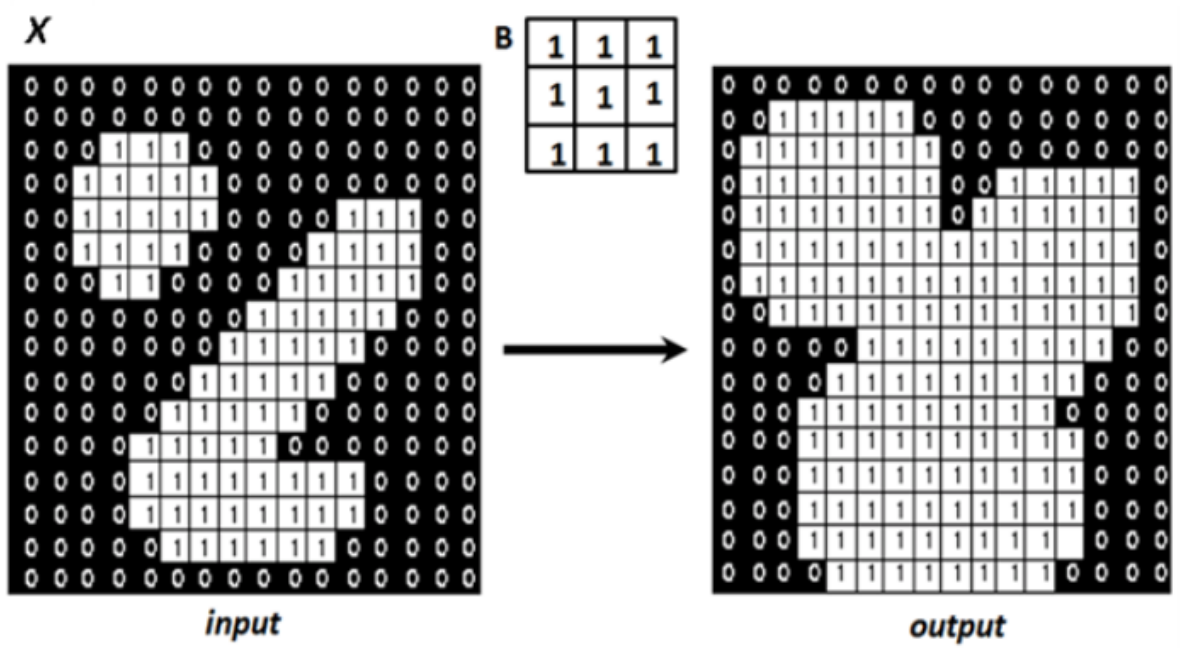


Figura 2.2 Aplicação da operação de dilatação sobre a imagem X utilizando o estruturante B. Fonte: `http://cse19-iiith.vlabs.ac.in/theory.php?exp=morph`.

2.1.2 Erosão

A erosão consiste em uma operação morfológica onde o valor do pixel é substituído pelo valor mínimo entre a vizinhança definida por um elemento estruturante. A operação de erosão diminui o tamanho dos objetos da imagem e, consequentemente, aumenta os espaços entre eles. A Figura 2.3 mostra o resultado da operação de erosão sobre a imagem X utilizando o elemento estruturante quadrado 3x3 B. Percebe-se que os objetos encolhem e a distância entre eles aumenta.

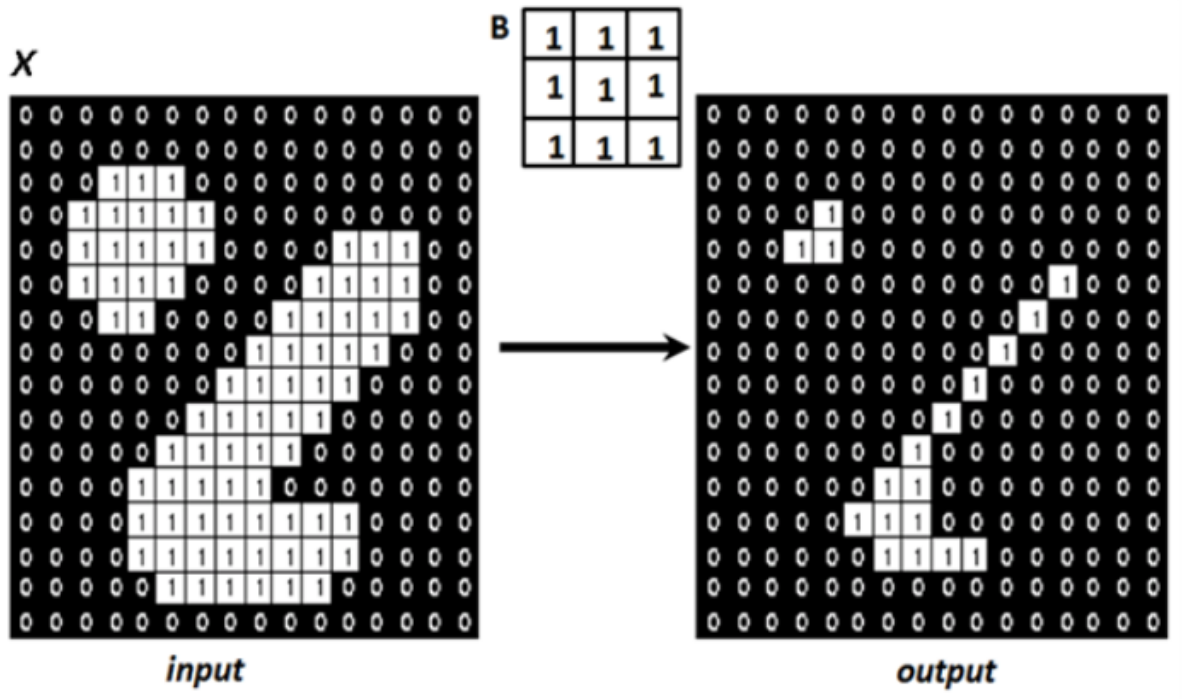


Figura 2.3 Aplicação da operação de erosão sobre a imagem X utilizando o estruturante B. Fonte: `http://cse19-iiith.vlabs.ac.in/theory.php?exp=morph`.

2.1.3 Abertura

A abertura consiste na aplicação da operação de erosão seguida da aplicação da operação de dilatação. A abertura suaviza a borda de objetos, quebra objetos estreitos e remove protuberâncias finas das bordas dos objetos. A Figura 2.4 mostra o resultado da operação de abertura sobre a imagem X utilizando o elemento estruturante quadrado 3x3 B. Observa-se que as bordas dos objetos foram “podadas”.

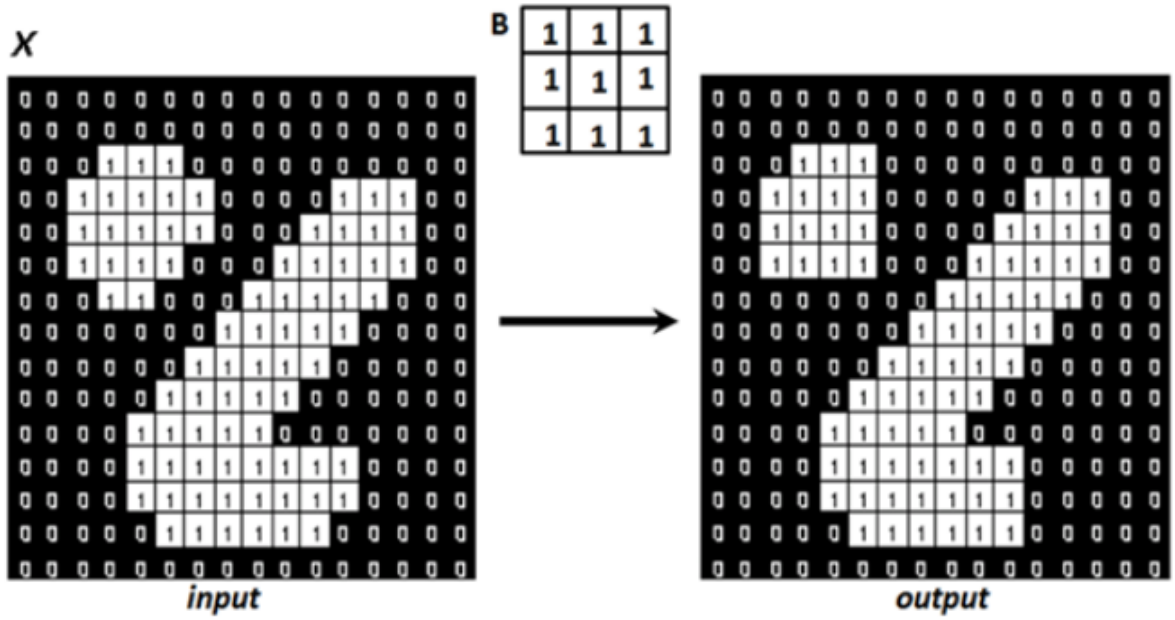


Figura 2.4 Aplicação da operação de abertura sobre a imagem X utilizando o estruturante B. Fonte: `http://cse19-iiith.vlabs.ac.in/theory.php?exp=morph`.

2.1.4 Fechamento

No fechamento ocorre o inverso da abertura, ou seja, primeiro é aplicada a dilatação e em seguida é aplicada a erosão. A operação de fechamento suaviza objetos inserindo pixels em suas bordas, elimina espaços estreitos, diminui entradas e elimina buracos menores que o elemento estruturante. A Figura 2.5 mostra o resultado da operação de abertura sobre a imagem X utilizando o elemento estruturante quadrado 3x3 B. Observa-se que as bordas dos objetos foram suavizadas, buracos e espaços foram preenchidos e a entrada do objeto maior foi diminuída.

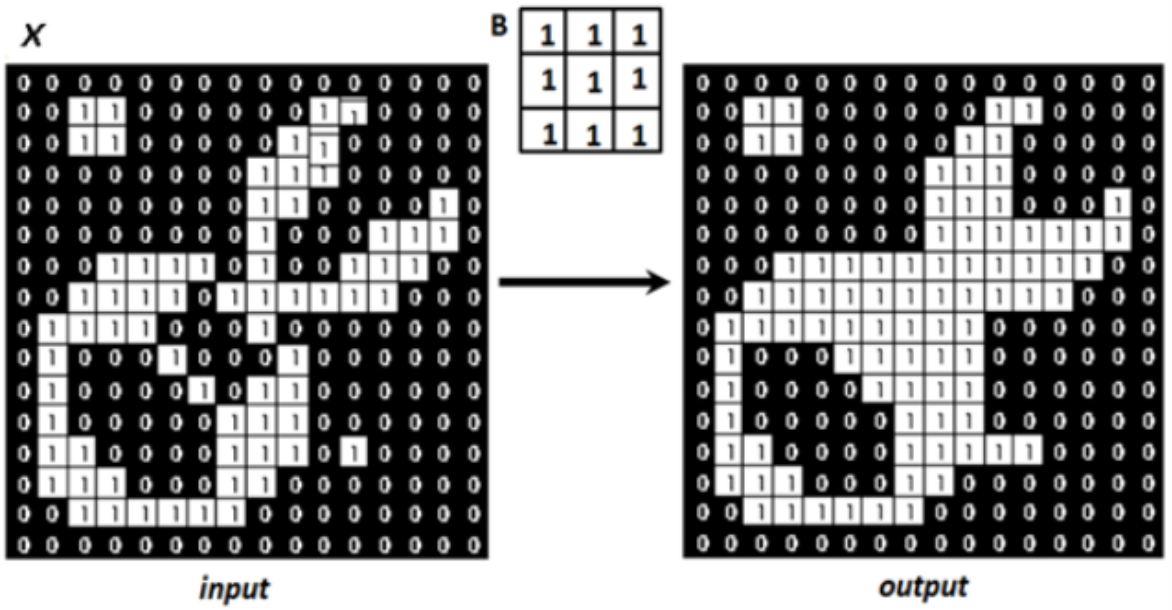


Figura 2.5 Aplicação da operação de fechamento sobre a imagem X utilizando o estruturante B. Fonte: `http://cse19-iiith.vlabs.ac.in/theory.php?exp=morph`.

2.2 Algoritmo de Verificação de Severidade de EMD

O algoritmo segue o fluxo apresentado na Figura 2.6. Para verificar a severidade do EMD, é necessário encontrar três estruturas da retina: o disco óptico, a fóvea e os exsudatos. A detecção do disco óptico é essencial pois a localização da fóvea é baseada na posição do centróide do disco óptico. A detecção da fóvea é necessária para a determinação da posição dos exsudatos em relação à fóvea, e com isso determinar a severidade do EMD. O algoritmo combina várias técnicas de processamento de imagem, em sua maioria operações morfológicas, como, por exemplo, abertura, fechamento e dilatação [RR92].

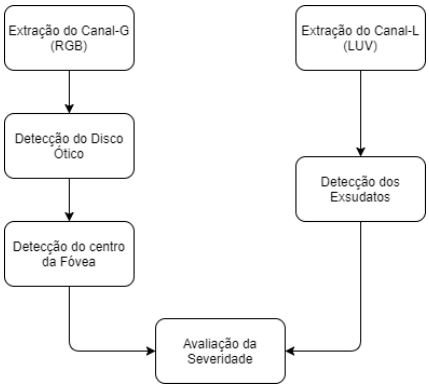


Figura 2.6 Fluxograma geral do algoritmo de verificação de severidade de EMD.

2.2.1 Detecção do Disco Óptico

A detecção do disco óptico segue quatro passos, que são ilustrados na Figura 2.7. O primeiro passo consiste em aplicar abertura e fechamento *Top-Hat*. A abertura *Top-Hat* consiste em aplicar abertura da imagem e subtrair o resultado da abertura da imagem original. O fechamento *Top-Hat* é semelhante, mas utiliza o fechamento em vez da abertura [DL03]. O resultado da abertura é adicionado ao canal-G da imagem e o resultado do fechamento é subtraído da mesma imagem. Esse processo tem o objetivo de destacar as partes mais claras da imagem (Figura 2.7.a). Na implementação em C, antes de executar a abertura e o fechamento, é passado um *threshold* para deixar o fundo da imagem preto. Caso o valor do pixel fosse menor que 14 em todos os canais, esse pixel receberia o valor 0 em todos os canais.

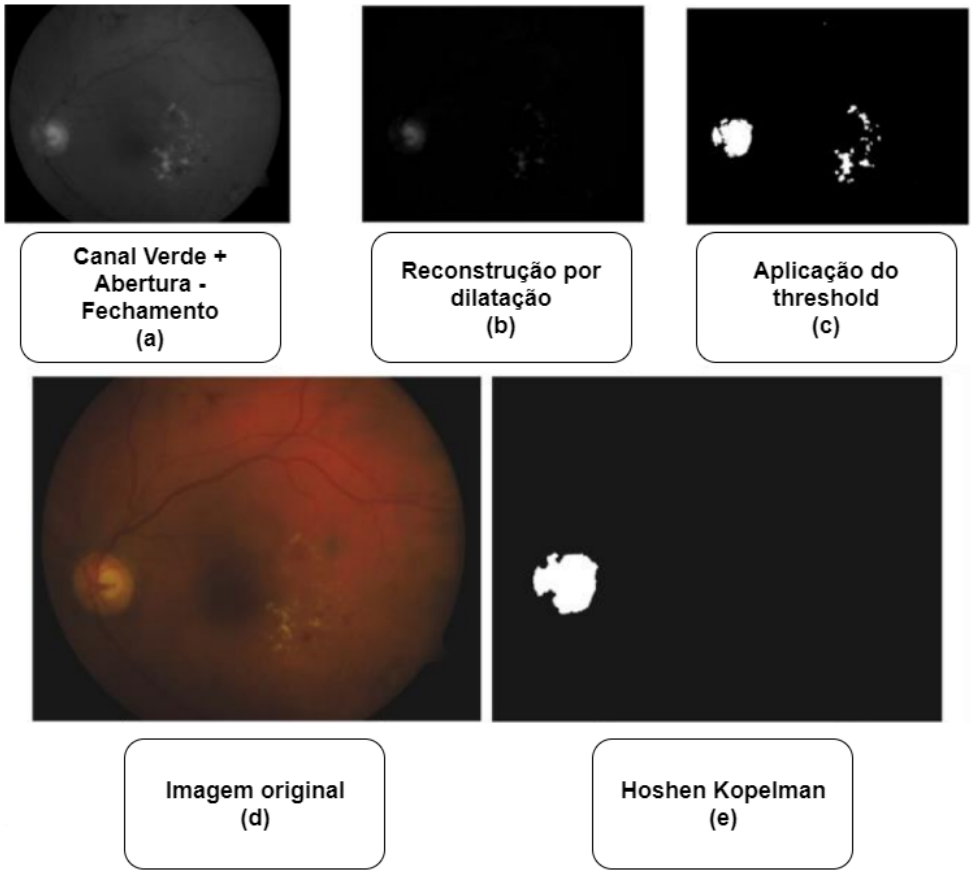


Figura 2.7 Imagens resultantes de cada passo do algoritmo de detecção de disco óptico.

O passo seguinte é encontrar os mínimos regionais que atuarão como máscara para uma reconstrução por dilatação. Foi usado um estruturante circular de tamanho 19 por 19 pixels, como mostrado na Figura 2.8.

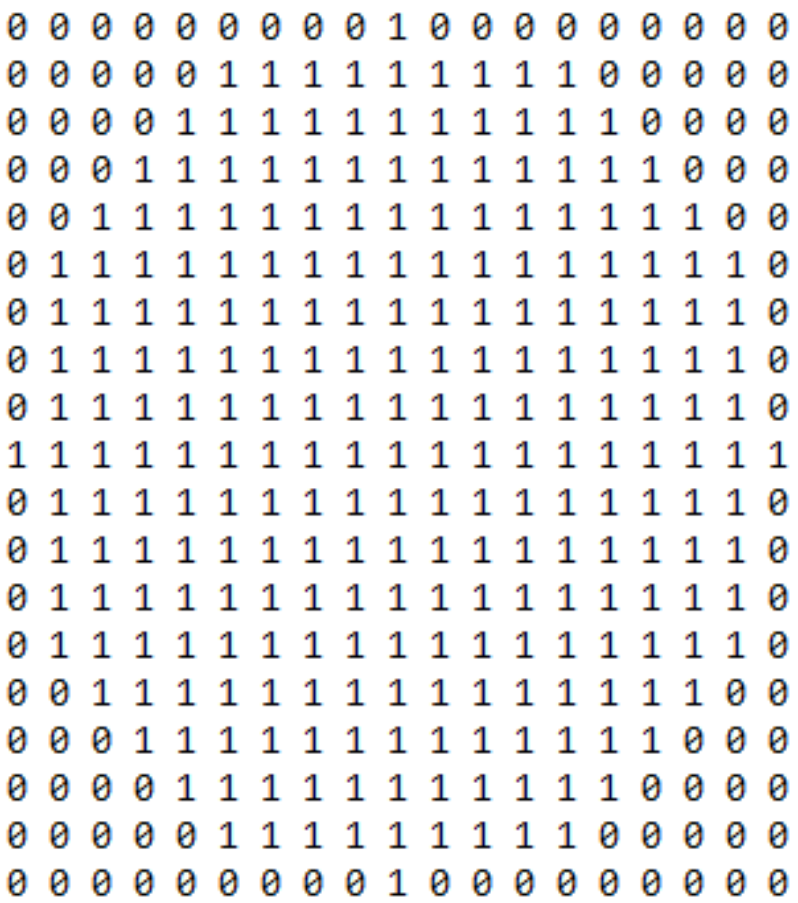


Figura 2.8 Estruturante circular com diâmetro de 19 pixels.

A reconstrução por dilatação é feita sobre a imagem resultante do processo de abertura e fechamento *Top-Hat* . Neste processo foi usado um estruturante em cruz 3 por 3 pixels, como pode ser observado na Figura 2.9. Esse processo resulta em uma imagem com o disco óptico e os exsudatos como pontos cinzas (Figura 2.7.b). Então é aplicada a transformada Hmáxima, que consiste em suprimir os pixels que tenham a intensidade menor que uma constante h [P.99]. Neste caso foi usado $h = 7$ (valor definido empiricamente). Depois foi aplicado um *threshold* = 8 para binarizar a imagem (Figura 2.7.c).

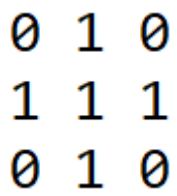


Figura 2.9 Estruturante em cruz com dimensões 3x3 pixels.

A próxima parte do algoritmo é usar o algoritmo de Hoshen Kopelman [HK76] para selecionar o maior conjunto de pixels claros, e então aplica-se uma dilatação, erosão e novamente

o algoritmo de Hoshen Kopelman para remover pontos negros dentro do disco óptico (Figura 2.7.e). Foi usado um estruturante em forma de diamante de tamanho 5 por 5 pixels, como mostrado na Figura 2.10. O centro do disco óptico é obtido pelo cálculo do centróide do conjunto de pixels resultantes.

```
0 1 1 1 0
1 1 1 1 1
1 1 1 1 1
0 1 1 1 0
0 0 1 0 0
```

Figura 2.10 Estruturante usado em forma de diamante de dimensões 5x5 pixels.

2.2.2 Detecção do Centro Fóvea

Para encontrar a região da fóvea, move-se o equivalente a 2,5 vezes o diâmetro do disco óptico em direção ao centro da retina. O diâmetro do disco óptico foi fixado empiricamente como sendo 80 pixels. Então é selecionada a região de interesse (ROI) alinhada ao centro do disco óptico (Figura 2.11.a). A ROI foi definida empiricamente como tendo 100x100 pixels. O próximo passo é aplicar operador de mínimo local e reconstrução morfológica por dilatação para remover os exsudatos. Neste passo é usado o mesmo estruturante em forma de diamante apresentado na Figura 2.10. Após a reconstrução, aplica-se operações de fechamento utilizando os elementos estruturantes apresentados na Figura 2.10 e Figura 2.8. O resultado pode ser observado na Figura 2.11.b. A área mais escura representa a fóvea e é segmentada pela aplicação de um *threshold*. Neste caso, o limiar usado foi valor do pixel mais escuro (Figura 2.11.c). O centro da fóvea é determinado pelo centróide da área segmentada no passo anterior (Figura 2.11.e).

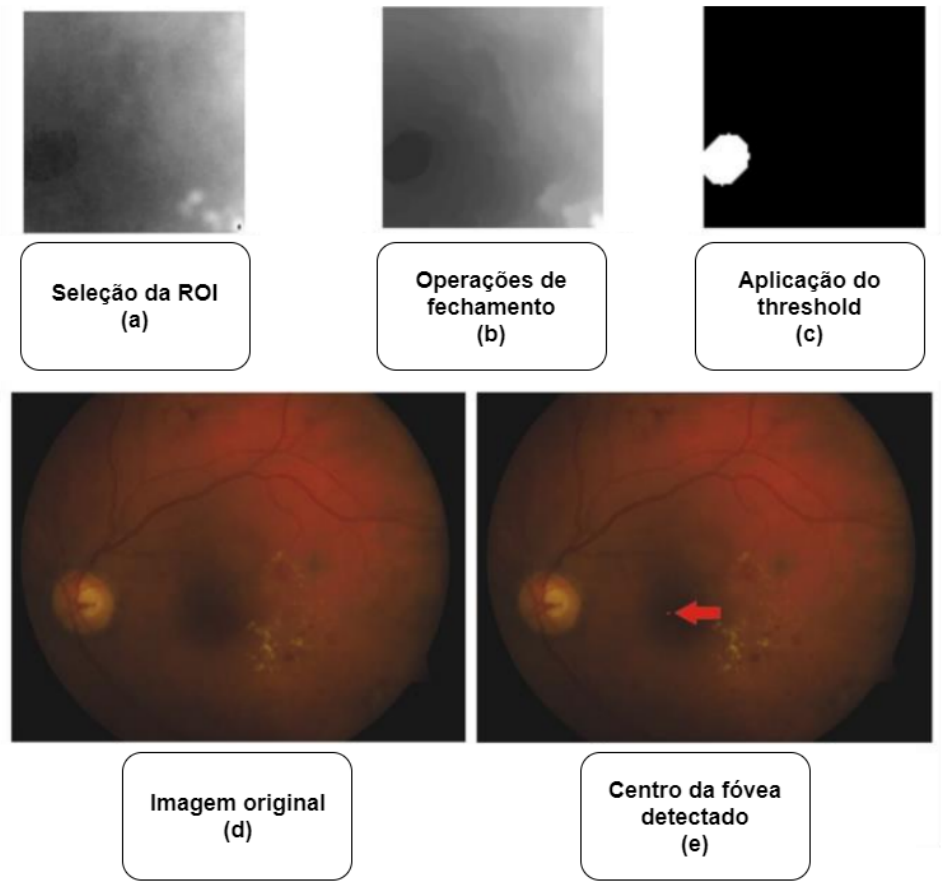


Figura 2.11 Imagens resultantes de cada passo do algoritmo de detecção do centro da fóvea.

2.2.3 Detecção dos Exsudatos

A detecção dos exsudatos é constituída por duas partes. Na primeira parte é feita uma detecção grosseira de exsudatos. Esse procedimento segue quase todos os passos da primeira parte do algoritmo de detecção do disco óptico, diferindo apenas em alguns pontos. O estruturante para usado para o mínimo regional foi em formato circular com diâmetro igual a 13 pixels (Figura 2.12) e na transformada H-máxima foi usado $h = 15$. Diferente do algoritmo de detecção do disco óptico, a detecção de exsudatos utiliza o espaço de cores LUV [Sch07], pois utiliza-se o canal L para detectar os exsudatos. O resultado pode ser visto na Figura 2.13.a.

```
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
```

Figura 2.12 Estruturante circular com diâmetro igual a 13 pixels.

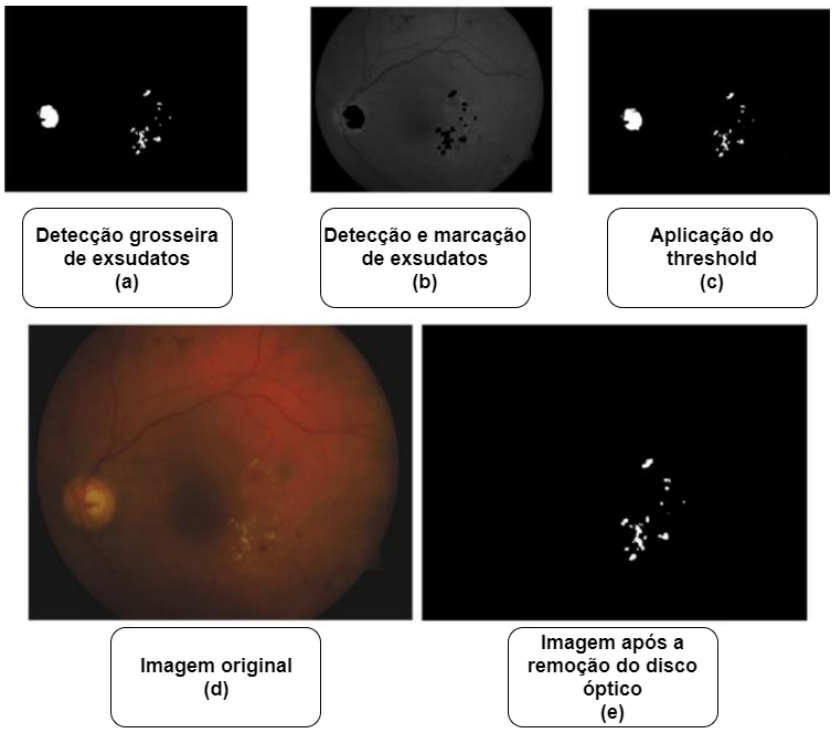


Figura 2.13 Imagens resultantes de cada passo do algoritmo de detecção de exsudatos.

Na segunda parte é feito um refinamento na detecção. Primeiro é aplicada uma dilatação utilizando um estruturante de dimensões 5x5 pixels em forma de diamante (Figura 2.10) e to-

dos os candidatos são marcados, substituindo os pixels do canal G da imagem RGB original por pixels pretos (Figura 2.13.b). A imagem resultante é usada em uma reconstrução por dilatação, usando o resultado da primeira parte (Figura 2.13.a) como máscara e com um elemento estruturante em circular 9x9 pixels (Figura 2.14). Subtraindo essas duas imagens e aplicando um *threshold* = 2, obtemos a Figura 2.13.c.

$$\begin{matrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$

Figura 2.14 Estruturante circular com diâmetro igual a 9 pixels.

Para remover os falsos exsudatos que permanecerem na imagem, são aplicadas dilatação e erosão com um elemento estruturante 3x3 em forma de L (Figura 2.15) . Para finalizar, o disco óptico é removido usando o resultado do algoritmo de detecção de disco óptico (Figura 2.13.e).

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{matrix}$$

Figura 2.15 Estruturante em forma de L com dimensões 3x3 pixels.

2.2.4 Avaliação da Severidade

Para avaliar a severidade da EMD, são seguidas as diretrizes descritas por Kanski [KB11] em termos da distância dos exsudatos à fóvea, mesmo que os exsudatos sejam muito pequenos. A área macular é dividida em 3 partes (central, intermediária e externa) como mostrado na Figura 2.16 e verificado o número de exsudatos em cada setor.

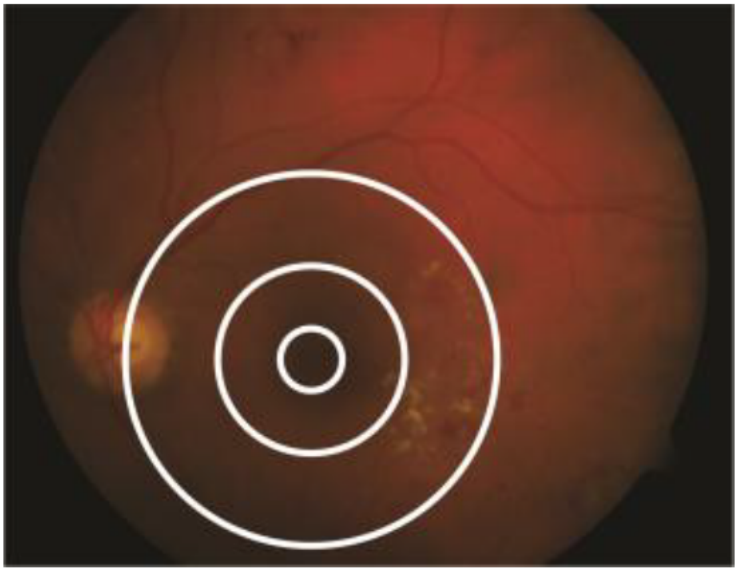


Figura 2.16 Setores da mácula. A menor circunferência delimita o setor central. A área entre a menor circunferência e a segunda menor circunferência corresponde ao setor intermediário. A área entre a maior circunferência e a circunferência intermediária corresponde ao setor externo da mácula.

2.3 RIFFA

Reusable Integration Framework for FPGA Accelerators (RIFFA) [JK15] é um framework simples para comunicação entre uma CPU hospedeira e um FPGA via um barramento PCI *Express*. Este *framework* requer um ambiente de trabalho com um PCIe habilitado e uma placa FPGA com um conector PCIe. RIFFA suporta Windows e Linux, Altera e Xilinx, com bibliotecas em C/C++, Python, MATLAB e Java.

A interface de software do RIFFA possui dois métodos principais: Enviar dados e receber dados. Essas funções podem ser utilizadas com auxílio de bibliotecas em C/C++, Python, MATLAB e Java. O driver suporta múltiplas FPGAs por sistema.

Do lado do hardware, o usuário acessa uma interface com sinais de transmissão e recebimento independentes. Não é necessário conhecimento sobre endereços de barramento, tamanho de buffer ou formato de pacotes PCIe. É necessário apenas enviar dados utilizando uma interface FIFO e receber os dados utilizando uma interface FIFO. RIFFA trabalha diretamente com o *endpoint* do PCIe e executa rápido o suficiente para saturar o *link* PCIe.

CAPÍTULO 3

Estado da Arte

A retinografia fluorescente [JJ13] é bastante utilizada no diagnóstico de retinopatia diabética. Este procedimento consiste em injetar uma substância fluorescente na veia do paciente. Esta substância é transportada até o olho através dos vasos sanguíneos. Com a chegada da substância no olho, uma câmera especial fotografa a retina, mostrando se alguma veia está obstruída, possui um vazamento ou está crescendo de forma anormal. Este procedimento necessita de uma infra estrutura para aplicar a substância contrastante na veia do paciente e de uma câmera especial, que além de ter um alto custo financeiro, é muito grande e de difícil transporte.

Outra técnica utilizada é a Tomografia de Coerência Óptica (OCT) [HSL$^+$91]. Essa técnica consiste em usar luz de baixa coerência para obter imagens de resolução micrométrica. Com essas imagens é possível fazer avaliação do disco óptico, da camada de fibras nervosas da retina, entre outras estruturas. Como a técnica anterior, esse procedimento necessita de equipamentos de alto custo financeiro e difícil transporte.

Ambas as técnicas citadas anteriormente geram imagens que necessitam da avaliação de um profissional para determinar a gravidade da retinopatia. Alguns trabalhos fazem a verificação de forma automática. As duas próximas seções deste capítulo irão abordar trabalhos de dois pesquisadores que desenvolveram técnicas de detecção automática de exudatos: Araka Sopharak [SUBW08] [SUB09] [SDU$^+$10] e Daniel Welfer[WSM10]. O capítulo se encerra com a síntese desses trabalhos.

3.1 Sopharak

Sopharak [SUBW08] [SUB09] [SDU$^+$10] propõe diferentes algoritmos de detecção de exsudatos. Eles empregam clusterização *fuzzy c-means*, operações morfológicas matemáticas e aprendizagem de máquina para remover vasos sanguíneos, detectar e remover o disco óptico e identificar exsudatos.

A implementação desse algoritmo foi feita completamente em software para computadores de mesa e demanda muito tempo para processar. O algoritmo que utiliza operações morfológicas matemáticas tem tempo de execução médio de 3 minutos em um processador Athlon 1.25 GHz, tendo a média de 1 minuto apenas na detecção do disco óptico. Já o tempo médio de execução do algoritmo que utiliza clusterização fuzzy c-means foi de 21 minutos ao ser executado no mesmo processador.

3.2 Welfer

Welfer[WSM10] propõe um método para detecção e localização de exsudatos. Seu trabalho usa operações morfológica e propõe um método mais preciso para detectar o disco óptico baseado na árvore de vasos sanguíneos. Após localizar o disco óptico, a localização da fóvea é estimada.

Este método é mais preciso, porém adiciona um elevado custo computacional. O algoritmo proposto por Gabriel Silva e Tiago Melo é baseado no método de Welfer, mas algumas etapas foram simplificadas, sobretudo a etapa de detecção de exsudatos, diminuindo o tempo médio de execução.

3.3 Síntese dos Trabalhos

O foco principal de todos esses trabalhos é a precisão independente do custo computacional. Os algoritmos propostos por esses trabalhos são voltados para computadores com grande capacidade de processamento e tempos de execução de alguns minutos. O algoritmo implementado neste trabalho apresenta uma abordagem baseada no trabalho de Welfer, mas modifica e otimiza vários passos, melhorando a performance e demandando menos memória, além de manter a precisão. Durante as pesquisas realizadas não foram encontrados algoritmos voltados para a implementação em *hardware*.

CAPÍTULO 4

Sistema Desenvolvido

Neste capítulo será descrito o sistema desenvolvido neste trabalho. Na primeira seção serão explicados os experimentos realizados para medir o desempenho do algoritmo implementado puramente em software. Será explicado a metodologia utilizada e apresentados os resultados. A segunda seção aborda as considerações e decisões de implementação, explicando o motivo pelo qual a dilatação foi o passo escolhido para ser implementado em HDL e o pré-processamento das imagens necessário para o funcionamento do sistema. A terceira seção explica a arquitetura do sistema. A arquitetura da placa será descrita nesta seção, assim como o fluxo de dados entre o processador e o FPGA. A última seção descreve os módulos desenvolvidos neste trabalho. Serão descritos a arquitetura e o funcionamento do núcleo de dilatação, responsável pelas comparações dos pixels, do módulo de dilatação, responsável por instanciar o núcleo de dilatação e controlar suas entradas, e a unidade de controle, módulo responsável pela comunicação com o processador e instanciar módulos de dilatação.

4.1 Experimentos Iniciais

4.1.1 Metodologia

Para selecionar etapa do algoritmo que foi implementada em hardware, foram feitas medições de tempo em diferentes partes do código do algoritmo implementado em C. As medições foram divididas em duas etapas.

Na primeira etapa, o algoritmo foi dividido em três passos: Detecção do disco óptico, detecção do centro fóvea e detecção dos exsudatos. Foi aferido o tempo antes e depois de cada parte e o tempo de execução de cada parte foi calculado como a diferença entre os dois tempos. O programa foi executado com 25 imagens diferentes, obtidas do banco de dados DIARETDB1 [KP07]. O passo com o maior tempo de execução médio foi selecionado para a segunda etapa.

Na segunda etapa, a parte selecionada do algoritmo foi subdividida em etapas menores e os tempos de cada etapa foram medidos utilizando a mesma técnica usada na primeira etapa. O algoritmo foi rodado sobre as mesmas imagens da primeira etapa. O resultado das medições foram utilizados como base para a escolha de qual etapa seria implementada em SystemVerilog e executada no FPGA.

Todas as medições foram feitas na placa DE2i-150 da Terasic, que possui um processador *Intel® Atom™ Dual Core Processor N2600*. O algoritmo foi executado sobre 25 imagens diferentes retiradas do banco de dados DIARETDB1 [KP07]. As imagens originalmente possuíam a resolução de 1500x1152 pixels. Com o objetivo de diminuir o custo computacional, a resolução foi modificada para 625x480, mantendo a proporção.

4.1.2 Resultados

Ao executar a primeira etapa das medições, foram obtidos os resultados apresentados na Tabela 4.1.

Passo	Média(ms)	Desvio Padrão(ms)
Detecção do disco óptico	46696	5465
Detecção do centro da fóvea	347	32
Detecção dos exsudatos	25887	3916

Tabela 4.1 Resultado da medição do tempo de execução dos passos do algoritmo de verificação de severidade de EMD.

Considerando que a detecção do disco óptico é a etapa mais custosa, foi executada a segunda etapa de medições sobre ela. A detecção do disco foi dividida em oito etapas, sendo elas: (1) Escurecer o fundo da imagem, (2) encontrar os mínimos regionais, (3) reconstrução por dilatação, (4) subtração das imagens, (5) transformada H-máxima, (6) aplicação de threshold, (7) aplicação do algoritmo de Hoshen Kopelman e (8) cálculo do centróide.

O resultado das medições podem ser observados na Tabela 4.2.

Etapa	Média(ms)	Desvio Padrão(ms)
Escurecer Fundo	278	10
Mínimos Regionais	28654	5182
Reconstrução por dilatação	14328	3537
Subtração das imagens	22	8
Transformada H-máxima	3017	564
Threshold	19	6
Hoshen Kopelman	376	10
Centróide	30	9

Tabela 4.2 Resultado da medição do tempo de execução das etapas do algoritmo de detecção de disco óptico.

4.2 Considerações e Decisões de Implementação

4.2.1 Seleção do passo

Como foi possível observar nos resultados das medições (Tabela 4.2), o passo mais custoso é o cálculo dos mínimos regionais utilizado na detecção do disco óptico, então este seria o passo mais indicado a ser implementado em SystemVerilog, porém, o estruturante usado é muito grande (circular de diâmetro 19 mostrado na Figura 2.8) e a convolução necessita de muita memória para ser aplicada.

A Figura 4.1 representa os pixels considerados em um instante qualquer da convolução em um trecho aleatório da imagem. Para esta etapa da convolução acontecer, é necessário que o pixel destacado em vermelho seja recebido e que todas as linhas anteriores que possuem pelo

menos um pixel em amarelo estejam armazenadas. O elemento estruturante tem 19 pixels de diâmetro, então é preciso acumular 18 linhas para que não se perca dados durante a convolução. Para isso, seria necessário o uso de memória RAM, pois a placa não possui registradores suficientes para acumular essa quantidade de dados (300 kB).

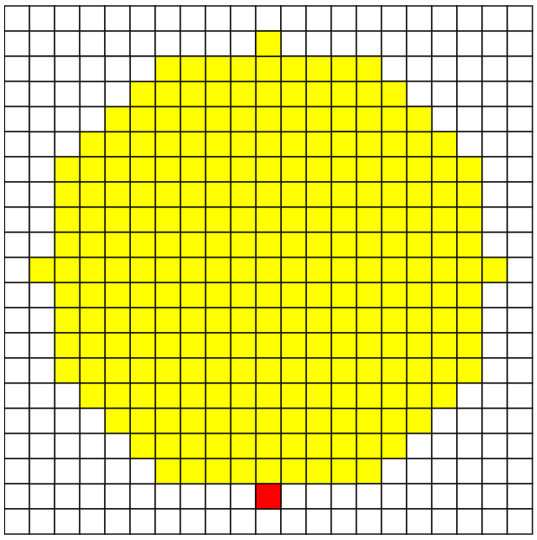


Figura 4.1 Representação de um instante da aplicação da convolução utilizando um elemento estruturante circular com diâmetro igual 19 pixels. O quadrado vermelho representa o último pixel que precisa ser recebido para a execução da cálculo.

A utilização de memória RAM iria demandar bastante tempo de aprendizado e não daria tempo de finalizar o trabalho, então foi implementada a dilatação usada na reconstrução por dilatação, que é o segundo passo mais custoso da detecção do disco óptico. O estruturante usado na dilatação tem forma de cruz com tamanho 3x3 pixels (Figura 2.9), sendo necessário acumular apenas duas linhas para executar a operação, como mostra a Figura 4.2, sendo assim possível implementar sem a utilização de memória RAM.

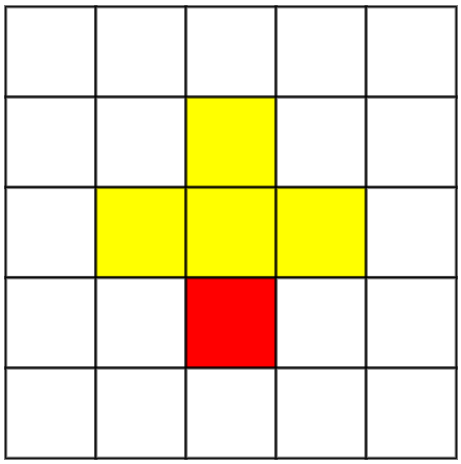


Figura 4.2 Representação de um instante da aplicação da convolução utilizando um elemento estruturante em formato de cruz com dimensões 3x3 pixels. O quadrado vermelho representa o último pixel que precisa ser recebido para a execução da cálculo.

Vale salientar que ao implementar a dilatação utilizando o estruturante em cruz, foi utilizado 70% do total de bits de memória da placa DE2i-150. Considerando que o número linhas acumuladas necessária para a implementação da convolução utilizando o estruturante circular com 19 pixels de diâmetro é 9 vezes maior que a implementação utilizando o estruturante em cruz, fica claro que a placa não suportaria o cálculo dos mínimos regionais sem a utilização de memória RAM.

4.2.2 Pré-processamento

A implementação da dilatação utilizada neste trabalho considera que serão enviados 8 pixels por ciclo de *clock* e que 8 pixels serão processados por ciclo. Por esse motivo, o número de pixels por linha das imagens de entrada deve ser múltiplo de 8. As imagens utilizadas no trabalho possuem 625 pixels por linha, portanto foi necessário adicionar 15 pixels pretos em cada linha das imagens após a leitura das imagens, como mostra a Figura 4.3. Após testes comparativos, foi constatado que essa mudança não afetou o resultado do algoritmo.

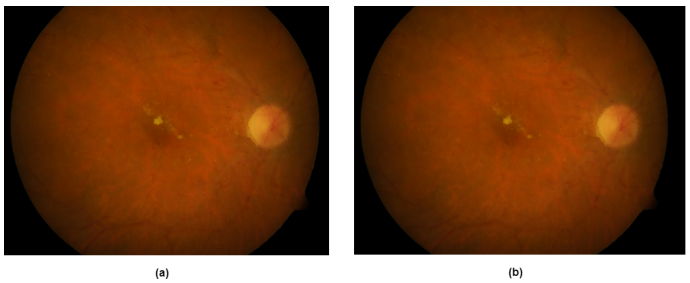


Figura 4.3 A imagem original (a) e a imagem após a inserção de 15 pixels pretos em cada linha (b)

4.3 Arquitetura do Sistema

O sistema utiliza a placa DE2i-150 da Terasic, que possui um processador *Intel Atom N2600* e um *FPGA Altera Cyclone IV EP4CGX150DF31* que se comunicam através de barramentos *PCI Express* . A Figura 4.4 descreve a arquitetura geral da placa.

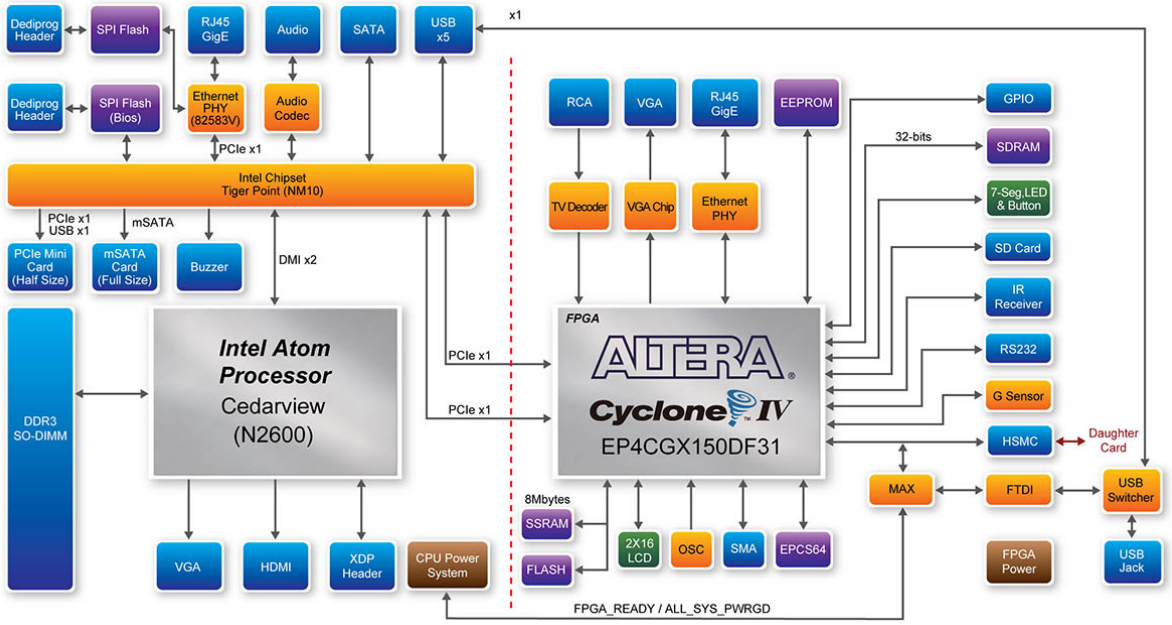


Figura 4.4 Diagrama de blocos que descreve a arquitetura da placa DE2i-150, da Terasic. Fonte: "`http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=11&No=529&PartNo=2`", acessado em 29/11/2017

A parte do algoritmo que foi implementada em C é executada no processador *Atom*, no qual foi instalado o sistema operacional *Windows 7*. A parte do algoritmo que foi implementada em *SystemVerilog* é executada no FPGA *Cyclone IV*.

A execução do programa se inicia em software com a leitura, escurecimento do fundo da imagem, aplicação da abertura e fechamento *Top-Hat* e o cálculo dos mínimos regionais. Após o cálculo dos mínimos regionais, é aplicada a reconstrução por dilatação, onde é feita a comunicação com o FPGA utilizando o RIFFA.

A imagem é enviada para o FPGA, que aplica a dilatação e retorna a imagem dilatada para o processador. O processador então compara a imagem resultante da dilatação com a a imagem que foi dilatada, se forem iguais, a reconstrução por dilatação chegou ao fim, caso contrário, a imagem resultante é enviada para o FPGA para ser dilatada. Esse processo é repetido até não haver diferença entre a imagem de entrada da dilatação e a imagem resultante. A Figura 4.5 mostra o fluxo de dados durante a reconstrução por dilatação.

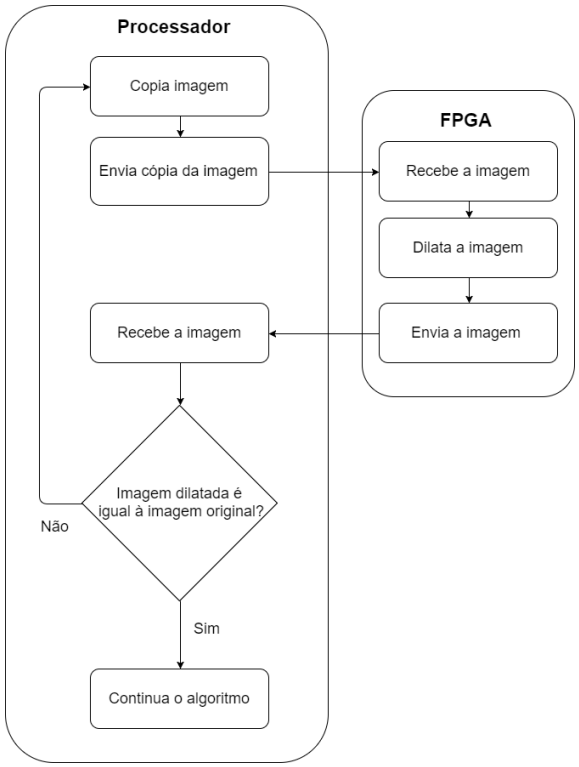


Figura 4.5 Fluxo de dados durante a reconstrução por dilatação do algoritmo de detecção de disco óptico do sistema desenvolvido.

4.4 Módulos desenvolvidos

Foram desenvolvidos 3 módulos em SystemVerilog: O núcleo de dilatação, o módulo de dilatação e a unidade de controle. A Figura 4.6 representa de maneira simplificada o fluxo de dados entre os módulos.

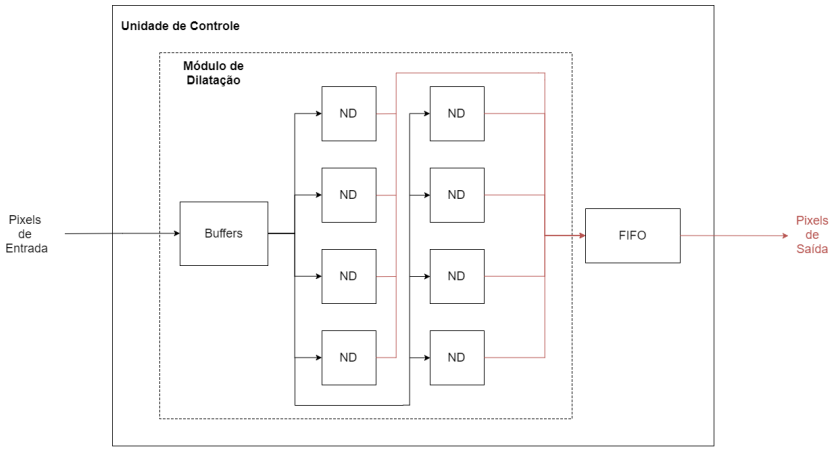


Figura 4.6 Diagrama que descreve o fluxo de dados entre os módulos. As setas pretas representam os pixels antes de serem processados. As linhas vermelhas representam os pixels após o processamento.

4.4.1 Núcleo de Dilatação

O núcleo da dilatação é o módulo responsável pela operações de comparação da dilatação . Este módulo recebe como entrada o pixel central e os quatro pixels vizinhos determinados pelo elemento estruturante, representados pela Figura 4.7, e retorna como saída o pixel com maior valor. Além de receber os pixels como entrada, existe um sinal de um bit que indica que o valor dos pixels são válidos e o *pipeline* deve ser iniciado.

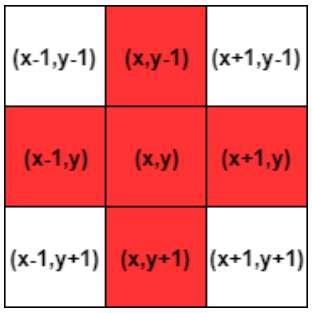


Figura 4.7 Pixel central e a sua 4-vizinhança. x e y representam o número da linha e o número da coluna na qual o pixel central se encontra, respectivamente.

Este módulo possui sete entradas e duas saídas, como mostra a Figura 4.8. O primeiro sinal de entrada é o *clock*, pois as operações são executadas na subida do *clock*. A segunda entrada é um sinal de um *bit* que indica quando o módulo deve começar o processamento e que os dados de entrada são válidos. As 5 entradas restantes são os pixels que serão considerados no processo, tendo 8 *bits* cada. As saídas dos módulos são o valor do pixel resultante e o sinal que indica que o processo foi finalizado, tendo 8 *bits* e 1 *bit*, respectivamente.

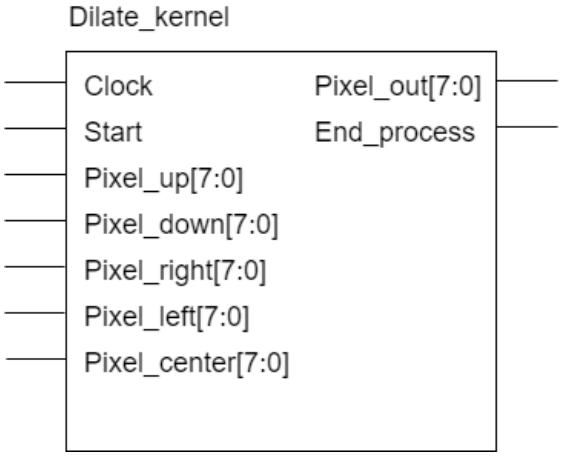


Figura 4.8 Bloco do núcleo de dilatação. Clock é a entrada para o *clock*, Start é um sinal de um *bit* que indica quando o módulo deve ser iniciado. Pixel_up, Pixel_down, Pixel_right, Pixel_left e Pixel_center são os pixels de entrada, que representam os pixels superior, inferior, da direita, da esquerda e o pixel central do elemento estruturante, respectivamente. Pixel_out é o pixel de saída do módulo. O sinal End_process indica quando o módulo finalizou a operação.

O núcleo da dilatação funciona como um pipeline de 5 estágios. Em cada ciclo é inserido um conjunto de pixels no pipeline, enquanto os conjuntos anteriores estão sendo processados nos diferentes estágios paralelamente. O primeiro estágio tem o simples objetivo de guardar a entrada em registradores para a próxima etapa. Se o sinal de início do *pipeline* estiver alto, os pixels são armazenados em registradores e é enviado um sinal indicando que o segundo estágio pode ser iniciado. Este sinal é importante pois ele indica que os dados que estão sendo escritos nos registradores são válidos. É necessário guardar o pixel central em registradores, pois o pixel central precisa acompanhar os dados no pipeline para evitar que seja feita a comparação com o pixel central de outra etapa. O segundo estágio faz duas comparações, o vizinho superior é comparado com o vizinho inferior e o vizinho da esquerda é comparado com o vizinho da direita. Os valores maiores das duas comparações e o pixel central são guardados em registradores para que a próxima etapa possa acessá-los. Além disso, é ativado o sinal para iniciar a próxima etapa. No terceiro estágio, é feita a comparação entre os resultados das comparações da etapa anterior. Como no segundo estágio, o maior valor e o pixel central são escritos em registradores e é ativado o sinal para iniciar a próxima etapa. O quarto estágio compara o valor resultante da comparação do terceiro estágio com o pixel central. O maior valor e o sinal para início da próxima etapa são gravados em registradores para que sejam acessados pela próxima etapa. No quinto estágio atribui o resultado do processo à saída e ativa o sinal que indica o final do processo, informando que o dado que está no output do módulo é válido.

A Figura 4.9 representa o funcionamento do *pipeline* do núcleo de dilação.

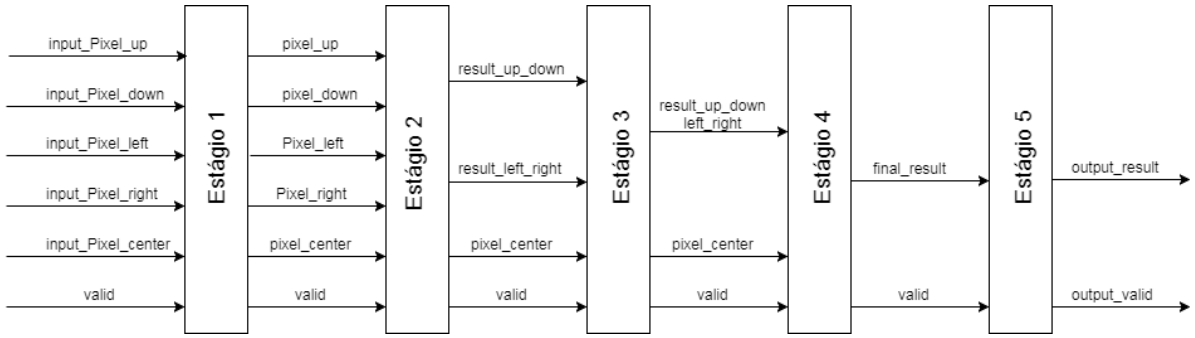


Figura 4.9 Representação do pipeline do núcleo de dilatação.

4.4.2 Dilatação

O módulo de dilatação tem o objetivo de instanciar o núcleo de dilatação e alimentá-lo com os pixels da imagem. Neste módulo também ocorre o tratamento de bordas, que consiste em considerar as bordas como tendo o valor 0, ou seja, quando o pixel estiver na borda, a vizinha que estiver fora da imagem recebe o valor 0.

Como foi explicado na subseção Seleção de passos da seção Considerações e Decisões de Projeto deste capítulo, é necessário acumular pixels da imagem antes de começar a dilatação da imagem. Para acumular, são usados dois vetores de registradores como filas de tamanho fixo que são preenchidas inicialmente com zeros. Ao inserir algo na fila, um elemento é removido, mantendo o tamanho da fila constante. A partir daqui essas filas serão chamadas de *buffers*

(Figura 4.10).

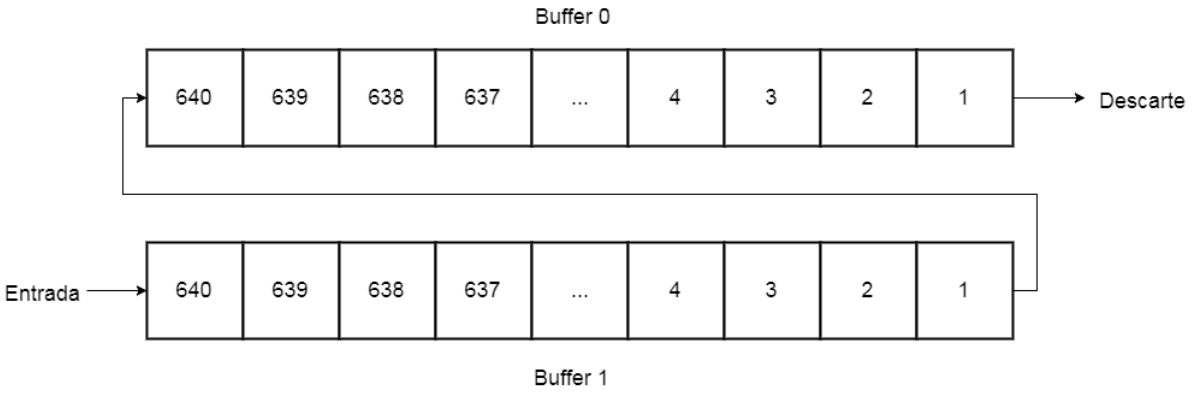


Figura 4.10 Esquema dos buffers usados para acumular as lihas da imagem.

Este módulo possui 4 entradas e 3 saídas, como mostra a Figura 4.11. As entradas consistem em um vetor de 64 *bits* e três sinais de um *bit* cada. Os pixels de entrada são recebidos pelo vetor de 64 *bits* (8 pixels de 8 *bits* cada). Os outros três sinais são o *clock*, o sinal de *reset* assíncrono e um sinal de validade que indica se os dados recebidos pelo vetor de 64 *bits* estão atualizados e podem ser processados. O sinal de validade é enviado para as instâncias do núcleo de dilatação para indicar a validade dos dados.

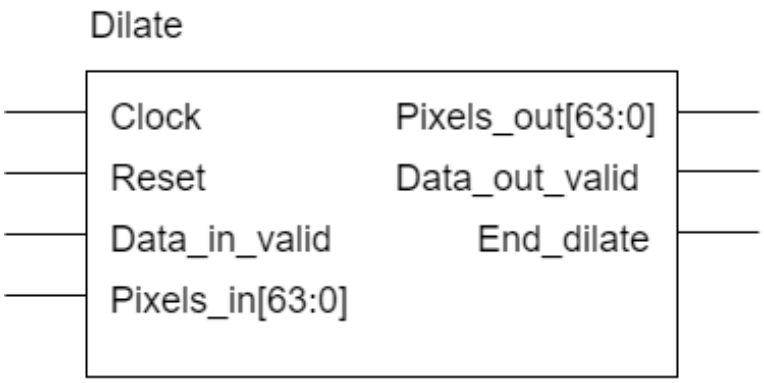


Figura 4.11 Bloco da dilatação. Clock é a entrada para o clock. Reset é um sinal de um bit do reset assíncrono. Data_in_valid é o sinal de validade dos dados. Pixel_in são os 8 pixels de entrada. Pixel_out são os 8 pixels de saída do módulo. Data_out_valid é o sinal de validade da saída do módulo. End_dilate indica o final do processamento do módulo.

Considerando que serão recebidos 8 pixels por vez através do RIFFA, o módulo de dilatação utiliza 8 instâncias do núcleo de dilatação para processar 8 pixel paralelamente e assim mantêm o fluxo de dados. A saída de cada instância do núcleo de dilatação é ligada diretamente à saída do módulo de dilatação e o sinal de término é usado com indicador de validade da saída, como mostra a Figura 4.12. Apenas um sinal de término é necessário, pois todas as instâncias

executam de forma sincronizada. Para simplificar a Figura 4.12, os registradores auxiliares foram omitidos e os *buffers* foram simplificados.

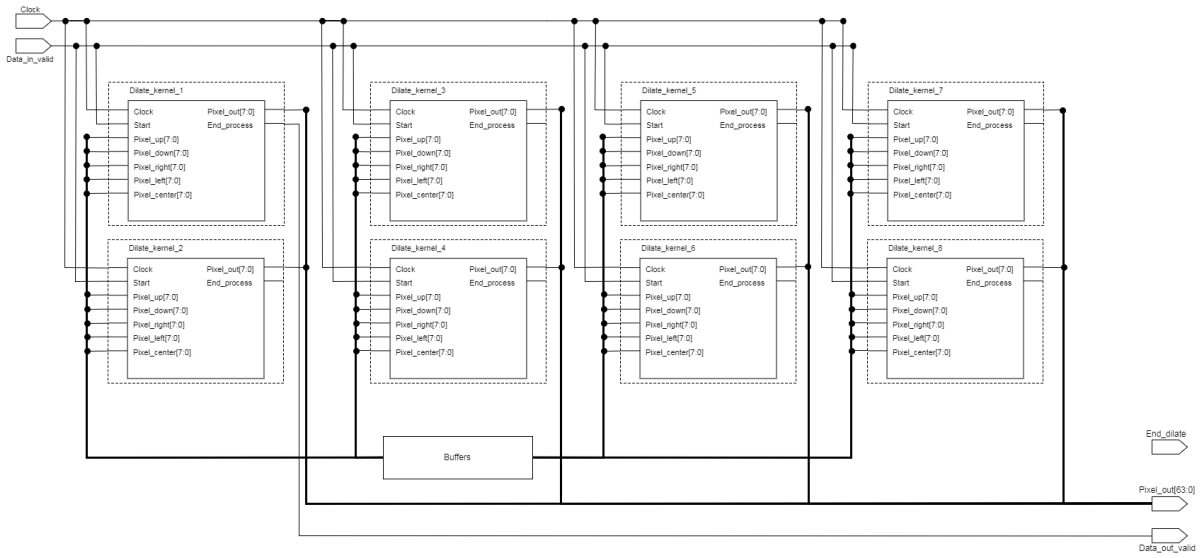


Figura 4.12 Diagrama de blocos do módulo de dilatação.

A Figura 4.13 representa os pixels utilizados em um instante qualquer da dilatação. Os pixels destacados com asterisco são os pixels centrais das operações. A primeira instância do núcleo de dilatação recebe o pixel identificado pelo número 10 como pixel central e os pixels identificados com os números 1, 11, 19 e 9 como os pixels que ficam acima, à direita, abaixo e à esqyerda do pixel central, respectivamente. A segunda instância do núcleo de dilatação recebe o pixel identificado pelo número 11 como pixel central e os pixels identificados com os números 2, 12, 20 e 10 como sua vizinhança. O restante das instâncias seguem o mesmo padrão.

Figura 4.13 Pixels utilizados em um instante aleatório da operação de dilatação.

Os pixel destacados em vermelho na Figura 4.13 formam o conjunto da vizinhança superior dos pixels centrais. Os pixels destacados em azul mais os pixels destacados em verde formam o conjunto dos pixels que ficam à esquerda do pixel central durante a dilatação. Os pixels destacados em verde mais os pixels destacados em amarelo formam o conjunto de pixels que ficam à direita do pixel central. Os pixels destacados em laranja formam o conjunto de pixels que se localizam abaixo dos seus respectivos pixels centrais. Os pixels marcados com um asterisco compõem o vetor de pixels centrais. Cada conjunto é armazenado em um vetor de registradores. Cada posição desses vetores servem como entrada para uma instância do núcleo de dilatação, assim como o *clock* e o sinal de validade.

É importante salientar que a imagem é recebida da direita para a esquerda, ou seja, o primeiro pixel recebido é o pixel mais a direita da primeira linha e o último pixel a ser recebido é o pixel mais à esquerda da última linha.

O módulo de dilatação funciona como uma máquina de estados com 7 estados e reset assíncrono, como mostra a Figura 4.14. Enquanto o sinal de *reset* estiver alto, as duas filas são preenchidos com zeros, todos os contadores são reiniciados e os registradores zerados.

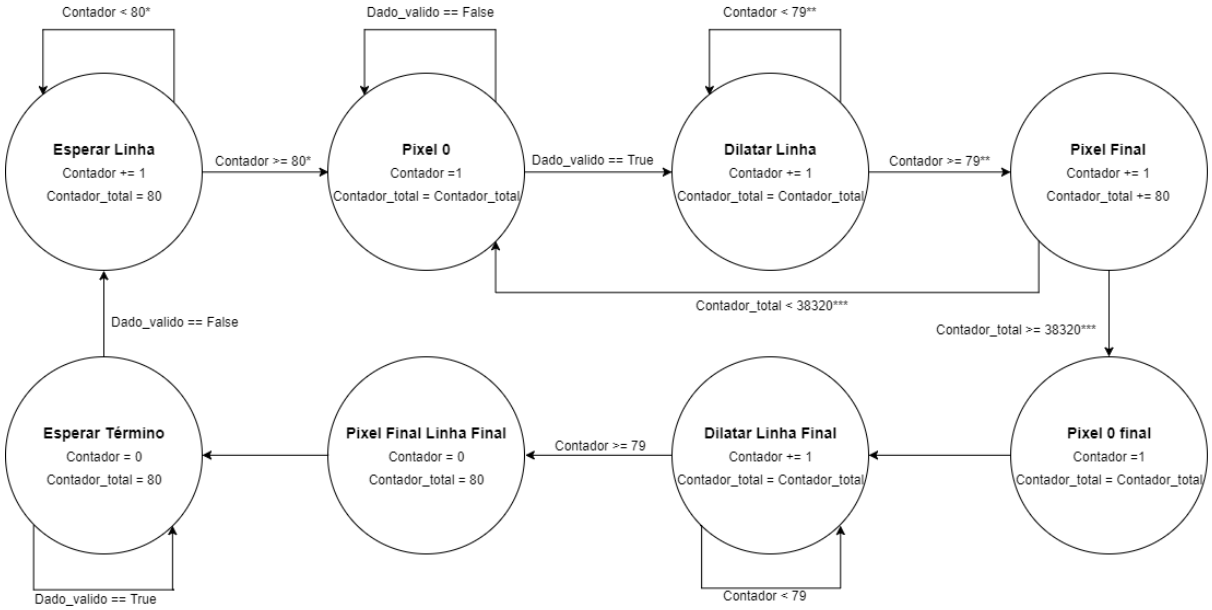


Figura 4.14 Máquina de estados do módulo de dilatação. * O valor 80 representa a quantidade de ciclos necessários para receber uma linha completa, considerando que são recebidos 8 pixels por vez e que a linha da imagem possui 640 pixels. ** O valor 79 é a quantidade de ciclos de clock necessários até o começo do processamento do último conjunto de pixels de uma linha desde o início do processamento desta linha. *** O valor 38320 é a quantidade de ciclos de clock até o começo do processamento do último conjunto de pixels da penúltima linha da imagem desde o início do processo de dilatação.

No primeiro estado é feito o recebimento da primeira linha. Enquanto os dados de entrada forem válidos, os oito pixels de entrada são inseridos em um dos *buffers* e um contador é incrementado. Ao inserir os pixels em um dos *buffers*, oito elementos da fila, que foi inicialmente preenchida com zeros, são removidos e inseridos no outro *buffer*, como ilustrado na Figura 4.10. A cada ciclo, um contador de operações por linha é incrementado. Esse processo é repetido durante toda o processo de dilatação.

A transição para o segundo estado ocorre quando o contador atinge o valor de 80, que é o número de inserções necessárias para completar uma linha, considerando que são inseridos 8 pixels por vez e cada linha possui 640 pixels. Esse valor é salvo em outro contador que guarda o número total de inserções. Ao fim do primeiro estado, a situação dos *buffers* será como ilustrado pela Figura 4.15.

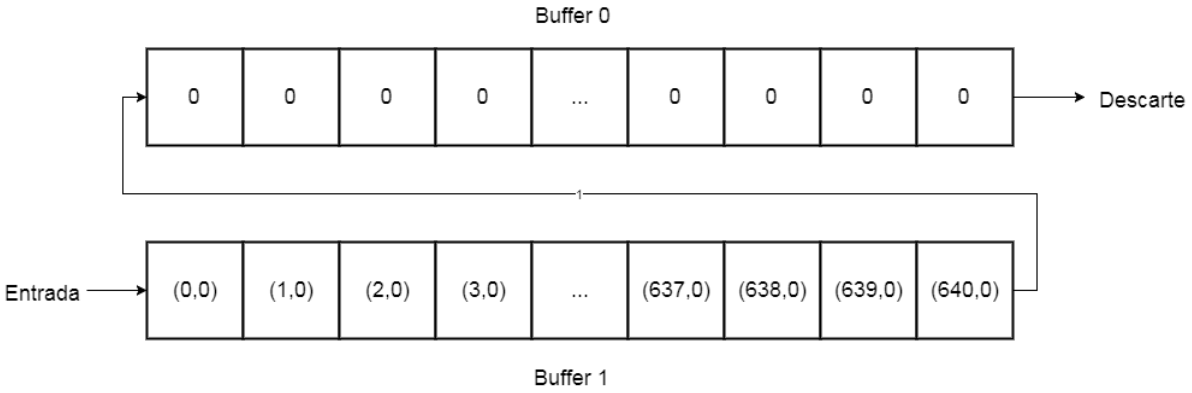


Figura 4.15 Estado dos buffers após acumular a primera linha. Os valores presentes no Buffer 1 representam a posição dos pixels na imagem original

No segundo estado, é feito o tratamento de borda do pixel mais à direita da linha. Neste caso, o pixel mais a direita entre os pixels centrais não possui vizinhança a sua direita, então é considerado um pixel de valor 0, como mostra a Figura 4.16.

Figura 4.16 Pixels utilizados no segundo estado da operação de dilatação. O pixel 18 representa a vizinhança à direita do pixel central 17. Por estar fora da imagem, ele receberá o valor 0.

A Figura 4.17 mostra quais as posições nos *buffers* dos pixels que compõem os conjuntos de entrada das instâncias do núcleo de dilatação. O esquema de cores utilizado na Figura 4.17 é o mesmo utilizado na Figura 4.13 para facilitar o entendimento. Os pixels destacados em laranja estão sendo recebidos pelo módulo de dilatação neste instante. Esses pixels compõem a vizinhança inferior dos pixels centrais. Os pixels centrais foram os primeiros a serem lidos no estado anterior, por esse motivo eles se encontram no início da fila do *buffer* 1, marcados

na Figura 4.17 com um asterisco. O pixel à esquerda de cada pixel central está localizado na posição imediatamente posterior na fila. O pixel à direita de cada pixel central está localizado na posição imediatamente anterior na fila, exceto o pixel à direita do pixel que está no início da fila, pois este é o pixel da borda direita da imagem, o que significa que seu vizinho à direita será o pixel auxiliar de valor 0. Os pixels destacados em vermelho compõem o conjunto de pixels que estão acima do pixel central. No início do estado anterior, esses pixels ocupavam exatamente as mesmas posições dos pixels centrais atuais e foram deslocados durante o processo, chegando no início da fila do *buffer* 0, então esses pixels estão exatamente acima dos pixels centrais na imagem original.

Figura 4.17 Representação de como os pixels são lidos dos buffers no segundo estado do módulo de dilatação.

Durante esse estado, o contador de inserções por linha é inicializado com 1 enquanto o contador do total de inserções não é alterado. Este estado dura apenas um ciclo de *clock*.

O terceiro estado é semelhante ao segundo, porém neste estado não há pixels nas bordas, não sendo necessário o uso do pixel auxiliar de valor 0. A Figura 4.18 mostra quais as posições nos *buffers* dos pixels que compõem os conjuntos de entrada das instâncias do núcleo de dilatação neste estado. Nota-se que em vez de utilizar o pixel auxiliar de valor 0, utiliza-se o pixel da posição 640 do *buffer 0* para compor o conjunto de pixels que ficam à direita dos pixels centrais, pois este pixel ocupava a posição imediatamente anterior ao pixel central mais à direita e foi deslocado para o final da fila do *buffer* 0.

Figura 4.18 Representação de como os pixels são lidos dos buffers no terceiro estado do módulo de dilatação.

A cada ciclo de *clock*, o contador de inserções por linha é incrementado enquanto o valor do contador do total de inserções é mantido. Quando o valor do contador de pixels por linha atinge 79, indicando que falta apenas uma inserção para completar uma linha da imagem, ocorre a transição para o quarto estado.

No quarto estado, é feito o tratamento de borda do pixel mais à esquerda da linha. Neste caso, o pixel mais à esquerda entre os pixels centrais não possui vizinhança a sua esquerda, então é considerado um pixel de valor 0, como mostra a Figura 4.19.

Figura 4.19 Pixels utilizados no quarto estado da operação de dilatação. O pixel 9 representa a vizinhança à esquerda do pixel central 10. Por estar fora da imagem, ele receberá o valor 0.

Neste estado, a posição dos pixels nos *buffers* é semelhante às posições do estado anterior, porém o pixel à esquerda do pixel central que está na borda esquerda da imagem é substituído

por um pixel auxiliar de valor 0, como mostrado na Figura 4.20.

Figura 4.20 Representação de como os pixels são lidos dos buffers no quarto estado do módulo de dilatação.

O contador de inserções por linha é incrementado e é somado 80 ao contador do total de inserções, pois esse é o número de inserções no *buffer* que foram feitas durante o processamento da linha atual. Esse estado dura apenas um ciclo de *clock*. No final do ciclo é verificado se a próxima linha a ser processada será a última, para isso é analisado o valor do contador do total de inserções. Sabendo que o número de pixels da imagem é igual à largura da imagem (640 pixels) multiplicada pela altura da imagem (480 pixels) e que o 8 pixels são processados por vez, temos que o número de operações necessárias para processar toda a imagem é de

$$\frac{640 \times 480}{8} = 38400. \tag{4.1}$$

Considerando que são feitas 80 operações por linha, conclui-se que o número de operações necessárias antes de começar a processar a última linha da imagem é

$$38400 - 80 = 38320. \tag{4.2}$$

Se o número total de inserções for menor que 38320, significa que a próxima linha a ser processada não será a última, então ocorre a transição para o segundo estado novamente, onde será processada a próxima linha. Caso o número de inserções não for menor que 38320, ocorre a transição para o quinto estado, onde ocorre o tratamento da borda inferior da imagem.

No quinto estado ocorre o tratamento da borda direita da última linha da imagem como, mostra a Figura 4.21.

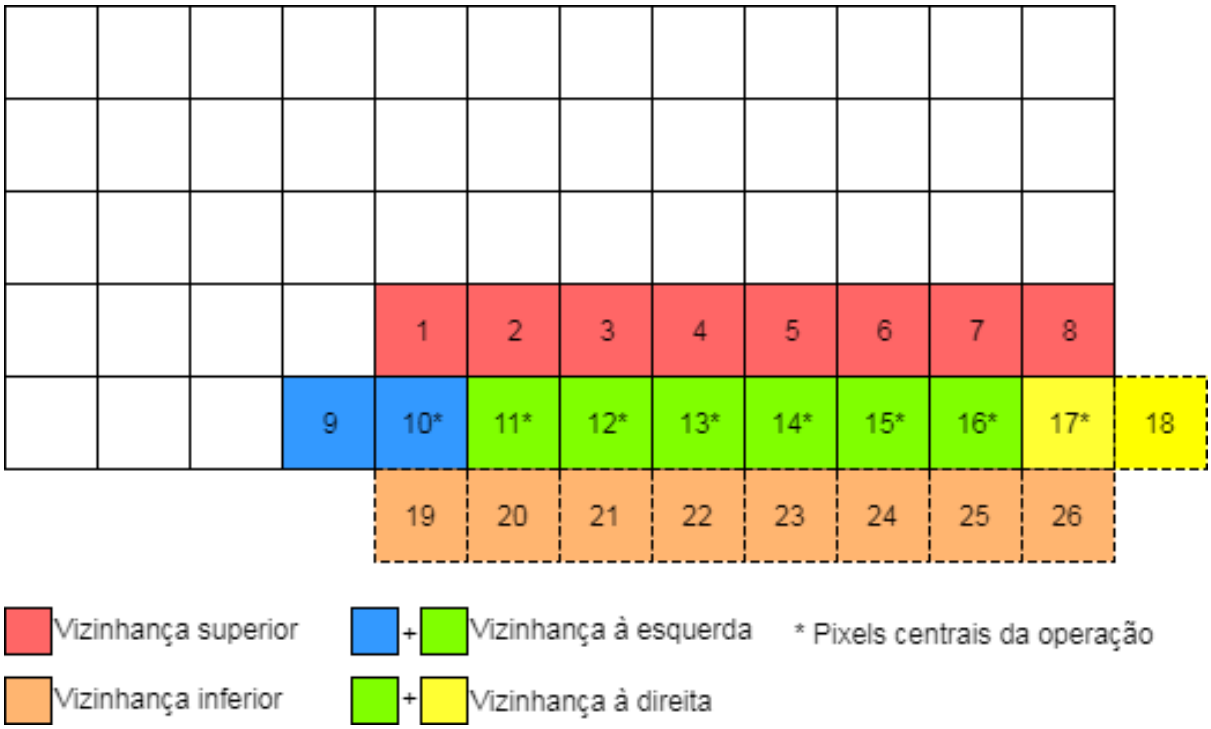


Figura 4.21 Pixels utilizados no quinto estado da operação de dilatação. O pixel 18 representa a vizinhança à esquerda do pixel central 17 e os pixels 19 até 26 são a vizinhança inferior dos pixels centrais 10 até 17. Por estarem fora da imagem, eles receberão o valor 0.

Este estado é semelhante ao segundo estado, diferindo apenas na vizinhança inferior dos pixels centrais. A vizinhança inferior é composta por pixels auxiliares de valor 0, como mostrado na Figura 4.22.

Figura 4.22 Representação de como os pixels são lidos dos buffers no quinto estado do módulo de dilatação.

O contador de inserções por linha é inicializado novamente com o valor 1 e o valor do contador do total de inserções é mantido. Neste momento, todos os pixels já foram recebidos,

então o sinal de entrada que indica a validade estará baixo. Por este motivo, neste estado o sinal de validade enviado às instâncias do núcleo de dilatação é fixado como alto em vez de utilizar o sinal de validade recebido como entrada do módulo de dilatação. Esse estado dura apenas um ciclo de *clock*, ocorrendo a transição para o próximo estado no mesmo ciclo de *clock*.

O sexto estado é semelhante ao terceiro estado, onde são tratados os pixels que não estão nas bordas laterais, porém neste estado existe o tratamento da borda inferior da imagem, como mostra a Figura 4.23.

Figura 4.23 Pixels utilizados no sexto estado da operação de dilatação. Os pixels 19 até 26 são a vizinhança inferior dos pixels centrais 10 até 17. Por estarem fora da imagem, eles receberão o valor 0.

A Figura 4.24 mostra quais as posições nos *buffers* dos pixels que compõem os conjuntos da entrada das instâncias do núcleo de dilatação neste estado. Observa-se que o conjunto da vizinhança inferior é composto por pixels auxiliares de valor 0.

Figura 4.24 Representação de como os pixels são lidos dos buffers no sexto estado do módulo de dilatação.

A cada ciclo de *clock*, o contador de inserções por linha é incrementado enquanto o valor do contador do total de inserções é mantido, assim como o sinal de validade dos dados. Quando o valor do contador de pixels por linha atinge 79, indicando que falta apenas uma inserção para completar uma linha da imagem, é feita a transição para o sétimo estado.

No sétimo estado é feito o tratamento da borda esquerda da última linha da imagem. Os últimos pixels da imagem começam a ser processados neste estado, como mostra a Figura 4.25.

Figura 4.25 Pixels utilizados no sétimo estado da operação de dilatação. O pixel 9 representa a vizinhança à esquerda do pixel central 10 e os pixels 19 até 26 são a vizinhança inferior dos pixels centrais 10 até 17. Por estarem fora da imagem, eles receberão o valor 0.

Este estado é semelhante ao estado quatro, diferindo apenas o conjunto de pixels que estão localizados abaixo do pixel central do elemento estruturante (Figura 4.26). Este estado envia o

último pixel para o núcleo de dilatação. A transição para o próximo ciclo é feito no primeiro ciclo.

Figura 4.26 Representação de como os pixels são lidos dos buffers no sétimo estado do módulo de dilatação.

O oitavo estado tem o objetivo de esperar o final do processamento das instâncias do núcleo de dilatação. Enquanto o sinal de validade das saídas das instâncias do núcleo de dilatação estiverem altos, o máquina de estados permanece neste estado. O sinal de saída que indica o término da dilatação é ativado, o sinal que indica a validade dos dados de saída é desativado e todos os registradores são reiniciados, assim como acontece quando o sinal de reset está ativado. Quando o sinal de validade das saídas das instâncias é desativado, ocorre a transição para o primeiro estado.

4.4.3 Unidade de Controle

A unidade de controle é o módulo de nível mais alto da arquitetura do sistema. Este módulo é responsável pela comunicação com o software e por instanciar o módulo de dilatação e controlar suas entradas e tratar suas saídas. Este módulo possui 10 entradas e 10 saídas, como mostrado na Figura 4.27.

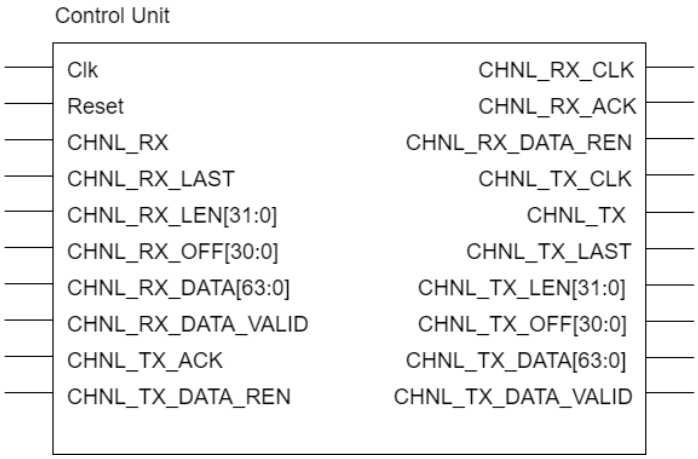


Figura 4.27 Bloco da unidade de controle. Clock é a entrada para o clock. Reset é um sinal de um bit do reset assíncrono. Todos os outros sinais são sinais de controle do RIFFA [oCSD17].

Esse módulo possui um sinal de *reset* e todos os outros sinais são usados na implementação do RIFFA e são detalhados no site oficial do RIFFA [oCSD17]. A unidade de controle utiliza apenas um canal para receber e enviar dados, por este motivo toda a imagem é recebida, só então o resultado é enviado de volta ao software. A imagem é recebida e enviada diretamente ao módulo de dilatação. A saída do módulo de dilatação é armazenada em um módulo FIFO, de onde é lida e enviada de volta ao *software* em C. O módulo FIFO utilizado foi baseado no módulo SCFIFO disponibilizado pela Intel [Int17]. O módulo FIFO utilizado possui 5 entradas e 4 saídas, como mostra a Figura 4.28. As entradas consistem no sinal de *clock*, 64 *bits* de dados, que são os 8 pixels de entradas, um sinal de requisição de escrita na fila, um sinal de requisição de leitura da fila e um sinal para limpar a fila. As saídas consistem em 64 *bits* de dados, que são 8 pixels de saída da fila, 16 *bits* que indicam quantos pixels estão na fila no momento, um sinal que indica que a fila está quase cheia e um sinal que indica que a fila está quase vazia. O módulo foi configurado para ativar o sinal que indica que a fila está quase vazia quando existir apenas um pixel na fila e ativar o sinal que indica que a fila está quase cheia quando faltar apenas um pixels para completar a fila. A fila foi configurada para ter o tamanho igual ao número de pixels da imagem.

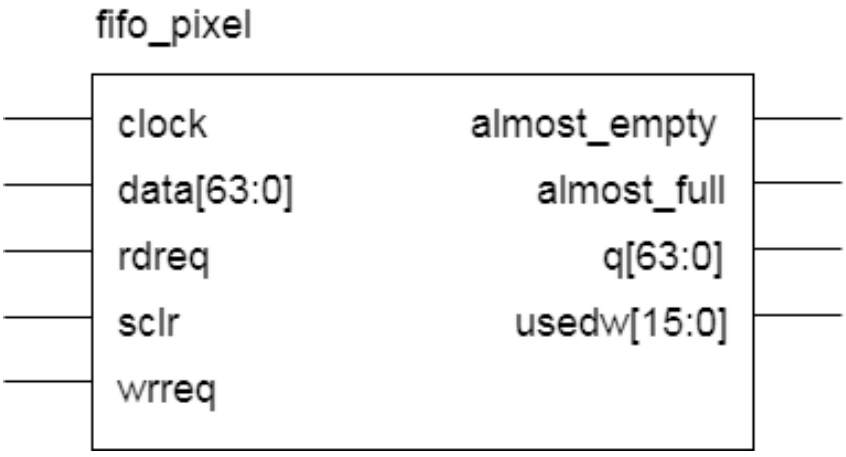


Figura 4.28 Bloco do módulo FIFO utilizado na unidade de controle. Clock é a entrada para o *clock*. Data é a entrada de dados de 64 *bits*, por onde são recebidos os 8 pixels provenientes da saída do módulo de dilatação. rdeq, sclr e wrreq são os sinais de requisição de escrita, limpeza da fila e requisição de escrita, respectivamente. almost_empty e almost_full são os sinais que indicam se a fila está quase cheia e quase vazia, respectivamente. q é a saída de 64 *bits* da fila, por onde são enviados os 8 pixels de saída da fila. usedw indica quantos pixels existem na fila.

A Figura 4.29 representa de forma simplificada como as instâncias do módulo de dilatação e o módulo FIFO se comunicam entre si e com as entradas e saídas da unidade de controle. Apenas a comunicação entre as instâncias do módulo de dilatação, o módulo FIFO e as entradas e saídas da unidade de controle são representadas na Figura 4.29, omitindo os sinais intermediários utilizados no controle das transições máquina de estado.

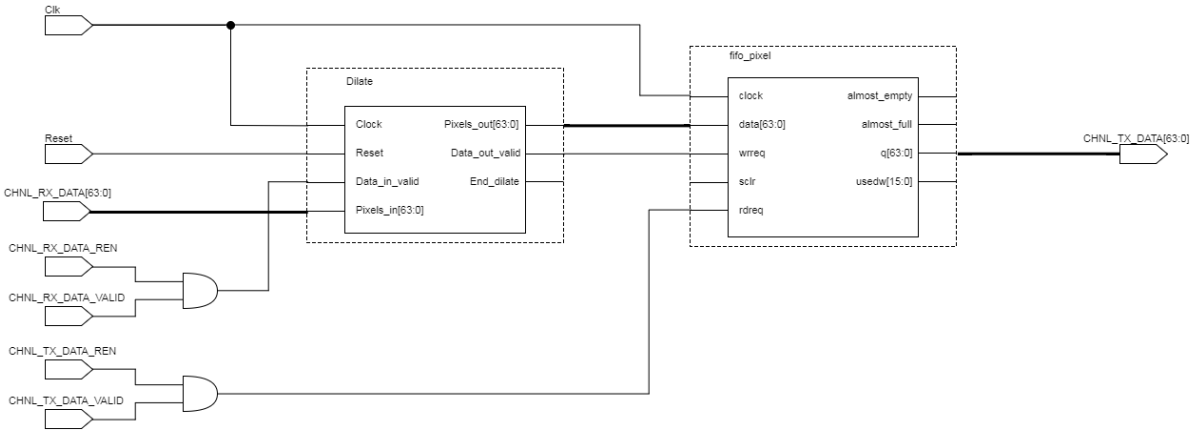


Figura 4.29 Diagrama de blocos simplificado da unidade de controle.

A unidade de controle funciona como uma máquina de estado de 5 estados com *reset* assíncrono, como mostra a Figura 4.30. Enquanto o sinal de *reset* estiver ativado, a fila é limpa e o processamento não é iniciado.

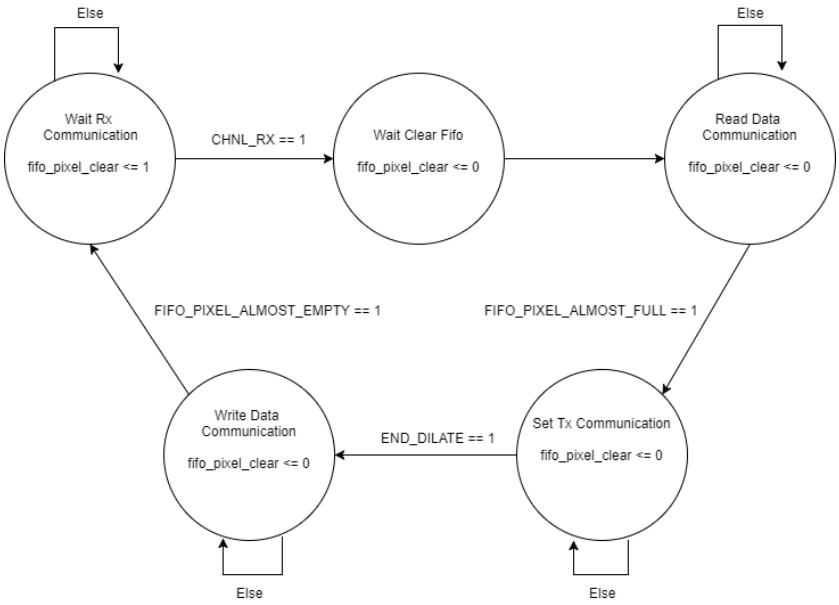


Figura 4.30 Maquina de estado da unidade de controle que foi implementada em SystemVerilog.

No primeiros estado, é esperado o sinal de início da comunicação. Durante esse estado, o sinal para limpar a fila é ativado. Quando o sinal de início da comunicação é recebido, ocorre a transição para o próximo estado.

O segundo estado tem o objetivo de esperar a fila ser limpa e desativar o sinal de limpar a fila. Este estado dura apenas um ciclo de *clock*, então a transição para o próximo estado ocorre no primeiro ciclo.

Durante o terceiro ciclo os pixels estão sendo recebidos e processados pelo módulo de dilatação. Os pixels resultantes, ou seja, a saída do módulo de dilatação, são armazenados na fila. Quando a fila é preenchida completamente, ocorre a transição para o próximo estado.

O quarto estado espera o término do processamento do módulo de dilatação e inicia a comunicação com o *software* em C, enviando o sinal de início da comunicação. A transição para o quinto estado ocorre quando o sinal de término do módulo de dilatação é ativado.

No quinto estado os pixels são lidos da fila e enviado através do RIFFA junto com o sinal de validade. Enquanto existir pixels na fila, a unidade de controle continua neste estado. Quando a fila estiver vazia, a unidade de controle volta ao primeiro estado, onde espera o início de uma nova comunicação.

Para validar o sistema, foram feitas simulações funcionais utilizando a ferramenta ModelSim. Foram usados os valores dos pixels de imagens como entrada e o resultado foi comparado à dilatação feita pelo programa escrito em C. Os resultados foram iguais.

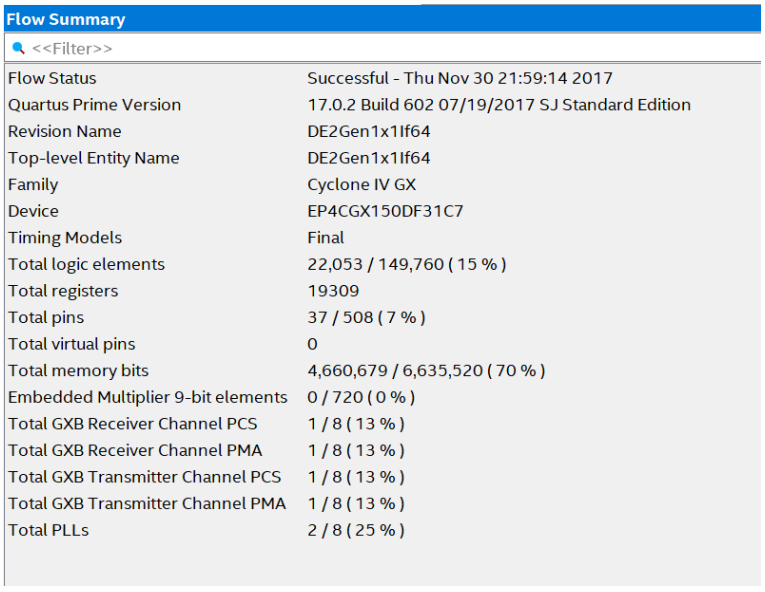
CAPÍTULO 5

Resultados e Discussão

Neste capítulo será feita uma análise breve do sistema e será explicado como os testes de desempenho do sistema foram feitos. Na primeira seção é feita uma análise breve sobre os recursos da placa DE2i-150 que foram utilizados pelo sistema. A segunda seção descreve os testes de desempenho do sistema, explicando qual foi a metodologia utilizada e apresentando os resultados, comparando os sistemas desenvolvidos neste trabalho com a implementação do algoritmo-base em *software*.

5.1 Análise

Os módulos desenvolvidos utilizaram um total de 22053 elementos lógicos, 15% da quantidade disponibilizada pela placa, 19309 registradores e 4660679 bits de memória, 70% da memória disponível na placa. A Figura 5.1 resume as estatísticas de síntese do projeto geradas pela ferramenta *Intel® Quartus® Prime*.



Flow Summary

<<Filter>>

Flow Status	Successful - Thu Nov 30 21:59:14 2017
Quartus Prime Version	17.0.2 Build 602 07/19/2017 SJ Standard Edition
Revision Name	DE2Gen1x1If64
Top-level Entity Name	DE2Gen1x1If64
Family	Cyclone IV GX
Device	EP4CGX150DF31C7
Timing Models	Final
Total logic elements	22,053 / 149,760 (15 %)
Total registers	19309
Total pins	37 / 508 (7 %)
Total virtual pins	0
Total memory bits	4,660,679 / 6,635,520 (70 %)
Embedded Multiplier 9-bit elements	0 / 720 (0 %)
Total GXB Receiver Channel PCS	1 / 8 (13 %)
Total GXB Receiver Channel PMA	1 / 8 (13 %)
Total GXB Transmitter Channel PCS	1 / 8 (13 %)
Total GXB Transmitter Channel PMA	1 / 8 (13 %)
Total PLLs	2 / 8 (25 %)

Figura 5.1 Estatísticas de síntese do projeto geradas pela ferramenta *Intel® Quartus® Prime*

Cerca de 680 linhas foram escritas em SystemVerilog, considerando a unidade de controle, o módulo de dilatação e o núcleo de dilatação.

5.2 Teste de Desempenho

5.2.1 Metodologia

Os testes foram realizados utilizando a placa DE2i-150. O programa em C, modificado para utilização do RIFFA, foi executado no processador *Intel® Atom™* sobre o sistema operacional Windows 7. A arquitetura implementada em SystemVerilog descrita no capítulo anterior foi executada no FPGA *Cyclone IV* da placa DE2i-150.

O sistema foi executado 25 vezes com 25 imagens diferentes. As imagens utilizadas foram as mesmas utilizadas nos experimentos iniciais, pois esse conjunto de imagens possuem todos os casos de severidade da doença (leve, moderado, grave e ausente). O tempo de cada execução foi medido e salvo. Foi feito o cálculo da média e do desvio padrão dos tempos de execução de todas as execuções.

Após feito todos os cálculos sobre essa versão do sistema, o *software* foi modificado para utilizar a dilatação implementada em FPGA na etapa de detecção dos exsudatos, pois a dilatação desta etapa utiliza o mesmo estruturante. As medições foram feitas nas mesmas condições das anteriores. Os tempos de execução foram salvos e foi calculada a média e o desvio padrão sobre o resultado.

5.2.2 Resultados

As Figuras 5.2. 5.3, 5.4 e 5.5 mostram exemplos de cada nível de severidade do EMD, sendo um caso leve, um moderado, um grave e um ausente, respectivamente. Em cada uma das figuras é mostrada a imagem original (a) e os setores centrais (b), intermediário (c) e externo (d) da mácula, destacados como setores circulares brancos. Os pixels pretos em cada setor são contados e usados como base para definir a severidade da doença, seguindo as diretrizes descritas por Kanski [KB11].

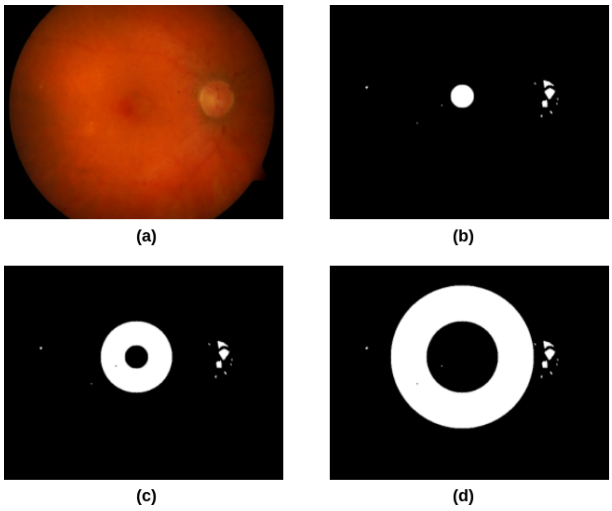


Figura 5.2 Caso de retinopatia com severidade leve. A imagem (a) é a imagem original do fundo de olho. A imagem (b) destaca o setor central da mácula, onde não foram encontrados exsudatos. A imagem (c) destaca o setor intermediário, onde foram contados 5 pixels negros, representando exsudatos. A imagem (d) destaca o setor externo, onde foram contados 5 pixels negros.

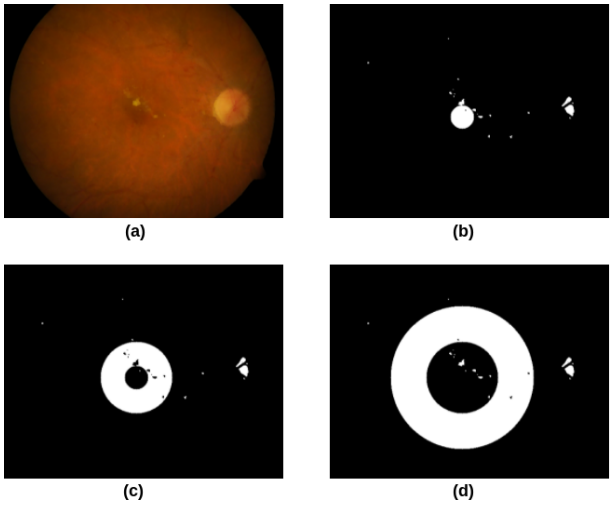


Figura 5.3 Caso de retinopatia com severidade moderada. A imagem (a) é a imagem original do fundo de olho. A imagem (b) destaca o setor central da mácula, onde foram contados 5 pixels negros, que representam os exsudatos. A imagem (c) destaca o setor intermediário, onde foram contados 258 pixels negros. A imagem (d) destaca o setor externo, onde foram contados 43 pixels negros.

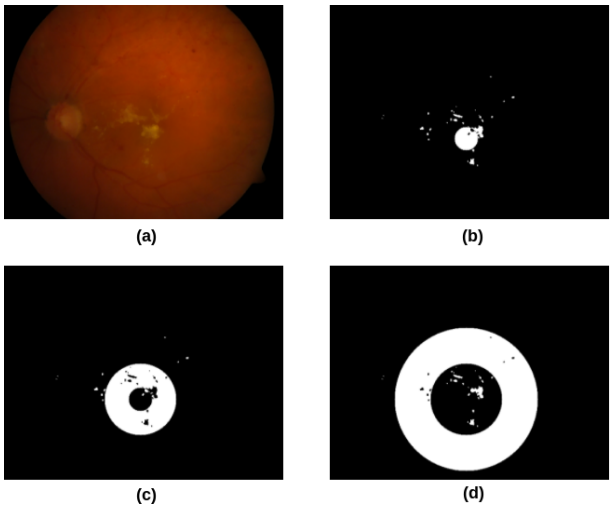


Figura 5.4 Caso de retinopatia com severidade grave. A imagem (a) é a imagem original do fundo de olho. A imagem (b) destaca o setor central da mácula, onde foram contados 68 pixels negros, que representam os exsudatos. A imagem (c) destaca o setor intermediário, onde foram contados 704 pixels negros. A imagem (d) destaca o setor externo, onde foram contados 130 pixels negros.

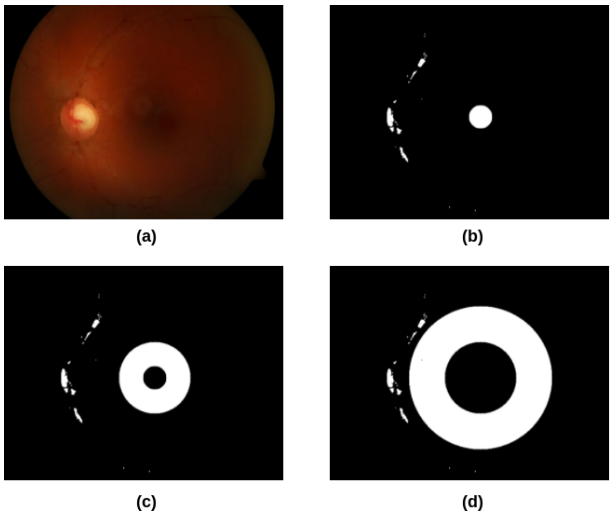


Figura 5.5 Caso de ausência de retinopatia. A imagem (a) é a imagem original do fundo de olho. Não foram encontrados pixels negros nos setores central (a) e intermediário (b). A imagem (d) destaca o setor externo, onde foram contados 3 pixels negros.

Os resultados do sistema desenvolvido neste trabalho foram exatamente iguais aos resultados da implementação do algoritmo puramente em software. Foram feitas comparações pixel a pixel entre as imagens resultantes da implementação em *software* e da implementação em *hardware/software* e não haviam diferenças entre elas. As localizações do disco óptico, do centro da fóvea e dos exsudatos foram iguais nos dois casos, consequentemente o diagnóstico gerado pelos dois sistemas foram iguais para todas as imagens.

O resultado dos testes de desempenho podem ser observados na Tabela 5.1. Na Tabela 5.1 estão presentes os tempos médios de execução do sistema implementado puramente em C, do sistema utilizando a implementação da dilatação em FPGA na detecção do disco óptico e do sistema que utiliza a implementação em FPGA na detecção do disco óptico e na detecção dos exsudatos.

Versão	**Média (ms)**	**Desvio Padrão (ms)**
Implementação em Software	72885	6995
Implementação em software e FPGA (Versão 1)	62576	5898
Implementação em software e FPGA (Versão 2)	57865	5928

Tabela 5.1 Tempos médios de execução da implementação original em software, da implementação do sistema utilizando FPGA apenas na detecção do disco óptico (Versão 1) e da implementação do sistema utilizando FPGA na detecção do disco óptico e na detecção dos exsudatos.

O tempo de execução médio da primeira versão do sistema desenvolvido é aproximadamente 14% menor que a versão original implementada completamente em software. A segunda versão do sistema desenvolvido neste trabalho apresenta uma melhora de aproximadamente 21% do tempo médio de execução em relação à implementação original e aproximadamente 8,5% em relação à primeira versão da arquitetura *hardware/software*.

CAPÍTULO 6

Conclusão e Trabalhos Futuros

Neste trabalho foi feita uma análise de desempenho sobre uma implementação do algoritmo de verificação da severidade de Edema Macular Diabético (EMD), desenvolvido por Gabriel Wanderley Albuquerque Silva e Thiago Soares de Melo, com o intuito de selecionar a etapa mais custosa e implementar esta etapa em SystemVerilog e desenvolver uma arquitetura *hardware/software* que implementa este algoritmo utilizando o RIFFA na comunicação entre o FPGA e o processador.

Foi constatado que a etapa mais custosa computacionalmente do algoritmo é o cálculo dos mínimos regionais executado na detecção do disco óptico, porém o FPGA utilizado, *Cyclone IV* presente na placa DE2i-150, não possui recursos suficientes para implementar este cálculo sem o uso de memória RAM. Por este motivo foi implementado a dilatação utilizada na reconstrução por dilatação durante a detecção do disco óptico.

Após a desenvolvimento do sistema, foram feitos testes comparativos entre a implementação original e a arquitetura *hardware/software*. O resultado dos experimentos mostraram a diminuição do tempo médio de execução em 14% quando a dilatação em FPGA é usada durante a detecção do disco óptico e em 21% quando a dilatação em FPGA é utilizada na detecção do disco óptico e na detecção dos exsudatos.. Esses resultados indica, que a utilização de FPGA aumenta o desempenho do sistema. Apesar de ser necessário pré-processar os dados de entrada e aumentar a complexidade de implementação, os benefícios da implementação de um arquitetura *hardware/software* são mais significantes que as desvantagens.

Para trabalhos futuros pretende-se implementar o cálculo de mínimos regionais utilizando memória RAM, pois por ser a etapa mais custosa do algoritmo, provavelmente haverá um ganho de desempenho maior que o apresentado pelo presente trabalho. Além do mínimo regional, outras partes do algoritmo serão implementadas em linguagem de descrição de *hardware*. Outro possível trabalho futuro é a utilização dos módulos desenvolvidos neste trabalho em outras aplicações.

Referências Bibliográficas

[DL03] Edward R. Dougherty and Roberto A. Lotufo. *Hands-on Morphological Image Processing (SPIE Tutorial Texts in Optical Engineering Vol. TT59)*, chapter 6. SPIE Publications, 2003.

[HK76] J. Hoshen and R. Kopelman. Percolation and cluster distribution. i. cluster multiple labeling technique and critical concentration algorithm. *Physical Review B*, 14(8):3438–3445, oct 1976.

[HSL+91] D. Huang, E. A. Swanson, C. P. Lin, J. S. Schuman, W. G. Stinson, W. Chang, M. R. Hee, T. Flotte, K. Gregory, C. A. Puliafito, and J. G. Fujimoto. Optical Coherence Tomography. *Science*, 254:1178–1181, November 1991.

[Int17] Intel. Scfifo and dcfifo ip cores user guide. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_fifo.pdf`, 2017. Acessado: 06/12/2017.

[JJ13] Maguire J.I. and Federman J.L. Intravenous fluorescein angiography. In *Tasman W, Jaeger EA, eds. Duane's Ophthalmology 2013 edition. Philadelphia, PA: Lippincott Williams & Wilkins; 2013:vol 3, chap 4*, 2013.

[JK15] Richmond D. Hogains M. Jacobsen, M. and R. Kastner. Riffa 2.1: A reusable integration framework for fpga accelerators. *ACM Transactions on Reconfigurable Technology and Systems*, September 2015.

[KB11] J.J. Kanski and B. Bowling. *Clinical Ophthalmology: A Systematic Approach*. Elsevier Health Sciences UK, 2011.

[KP07] Kalesnykiene V. Kamarainen J.-K. Lensu L. Sorri I. Raninen A. Voutilainen R. Uusitalo H. Kalviainen H. Kauppi, T. and J. Pietila. Diaretdb1 diabetic retinopathy database and evaluation protocol. *Medical Image Understanding and Analysis (MIUA)*, pages 61–65, 2007.

[oCSD17] University of California San Diego. Riffa A reusable integration framework for fpga accelerators. `http://riffa.ucsd.edu/node/3`, 2017. Acessado: 30/11/2017.

[Org17] World Health Organization. Priority eye diseases. `http://www.who.int/blindness/causes/priority/en/index.html`, 2017. Acessado: 29/11/2017.

[P.99] Soille P. *Morphological Image Analysis: Principles and Application*, pages 170–171. Springer-Verlag, 1999.

[RR92] Gonzalez R. and Woods R. *Digital Image Processing*, pages 618–519, 549. Pearson Prentice Hall, 1992.

[Sch07] János Schanda. *Colorimetry: Understanding the CIE System*, pages 61–64. Wiley-Interscience, 2007.

[SDU+10] Akara Sopharak, Matthew N. Dailey, Bunyarit Uyyanonvara, Sarah Barman, Tom Williamson, Khine Thet Nwe, and Yin Aye Moe. Machine learning approach to automatic exudate detection in retinal images from diabetic patients. *Journal of Modern Optics*, 57(2):124–135, jan 2010.

[SUB09] Akara Sopharak, Bunyarit Uyyanonvara, and Sarah Barman. Automatic exudate detection from non-dilated diabetic retinopathy retinal images using fuzzy c-means clustering. *Sensors*, 9(3):2148–2161, mar 2009.

[SUBW08] Akara Sopharak, Bunyarit Uyyanonvara, Sarah Barman, and Thomas H. Williamson. Automatic detection of diabetic retinopathy exudates from non-dilated retinal images using mathematical morphology methods. *Computerized Medical Imaging and Graphics*, 32(8):720–727, dec 2008.

[WSM10] Daniel Welfer, Jacob Scharcanski, and Diane Ruschel Marinho. A morphologic three-stage approach for detecting exudates in color eye fundus images. *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC 10*, 2010.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).