UNIVERSIDADE DE SÃO PAULO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO

JOVANDER DA SILVA FREITAS

Implementação de uma arquitetura para binarização de imagens em FPGA

São Carlos
2012

JOVANDER DA SILVA FREITAS

Implementação de uma arquitetura para binarização de imagens em FPGA¹

Dissertação apresentada ao Departamento de Engenharia Elétrica e Computação da Universidade de São Paulo, para obtenção do Título de Mestre em Ciências, Programa de Engenharia Elétrica.

Área de Concentração: Processamento de Sinais e Instrumentação

Orientador: Prof. Dr. Maximilian Luppe

**São Carlos
2012**

¹ Trata-se da versão corrigida da dissertação. A versão original se encontra disponível na EESC/USP que aloja o Programa de Pós-Graduação de Engenharia Elétrica.

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

**Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP**

Freitas, Jovander da Silva

F862i Implementação de uma arquitetura para binarização de imagens em FPGA. / Jovander da Silva Freitas ; orientador Maximilian Luppe. -- São Carlos, 2012.

Dissertação (Mestrado - Programa de Pós-Graduação em Engenharia Elétrica e Computação e Área de Concentração em Processamento de Sinais e Instrumentação)-- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. FPGA. 2. Binarização. 3. Processamento de imagens. 4. Otsu. I. Título.

FOLHA DE JULGAMENTO

Candidato: Bacharel JOVANDER DA SILVA FREITAS.

Título da dissertação: "Implementação de uma arquitetura para binarização de imagens em FPGA".

Data da defesa: 13/09/2012

Comissão Julgadora:

Resultado:

Prof. Dr. Maximilian Luppe (Orientador)
(Escola de Engenharia de São Carlos/EEESC)

Aprovado

Prof. Dr. Emerson Carlos Pedrino
(Universidade Federal de São Carlos/UFSCar)

Aprovado

Prof. Dr. Jorge Luiz e Silva
(Instituto de Ciências Matemáticas e de Computação/ICMC)

Aprovado

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica e Presidente da
Comissão de Pós-Graduação:
Prof. Titular Denis Vinicius Coury

DEDICATÓRIA

Aos meus pais, pela admiração e gratidão pela compreensão, carinho e incansável apoio ao longo do período de elaboração deste trabalho.

AGRADECIMENTOS

Ao amigo e Prof. Dr. Maximilian, que muito me apoiou e me auxiliou através de seu profundo conhecimento; a todos que direta ou indiretamente contribuíram para a realização deste trabalho.

EPÍGRAFE

“Ninguém é tão grande que não possa aprender, nem tão pequeno que não possa ensinar.”

Píndaro

RESUMO

FREITAS, Jovander da Silva. **Implementação de uma arquitetura para binarização de imagens em FPGA**. 2012. 76f. Dissertação (Mestrado) Departamento de Engenharia Elétrica e Computação, Universidade de São Paulo, São Carlos, 2012.

Em muitas aplicações de processamento de imagens é desejável converter imagens que estão em escala de cinza para imagens binárias, ou seja, em apenas dois níveis de intensidade. Para realizar essa tarefa de separação entre dois níveis é necessário que se calcule um valor de limiar, pois a partir dele determinamos quais *pixels* irão pertencer a um nível, normalmente o objeto de interesse, e quais pertencerão ao outro nível, ou ao fundo da imagem. Algumas aplicações exigem que se calcule esse valor de limiar em um tempo muito curto em relação à aquisição de imagem, principalmente quando ocorre uma variação muito alta de luminosidade na aquisição de uma imagem. Para suprir essa dificuldade de velocidade nas aplicações de processamento de imagem, uma alternativa seria o desenvolvimento de uma arquitetura dedicada que realize o cálculo do valor de limiar e binarize a imagem adquirida. O presente trabalho apresenta o desenvolvimento de uma arquitetura que realiza estas tarefas, implementada em circuitos reconfiguráveis do tipo FPGA. A validação da arquitetura foi obtida por meio da comparação dos resultados obtidos com a simulação da mesma na ferramenta Matlab. A arquitetura permite uma frequência máxima de clock de 84,52 MHz, o que permite a utilização da arquitetura em sistemas de tempo real, utilizando como fonte de imagem um vídeo composto ou uma câmera comum.

Palavras-Chaves: FPGA, Binarização, Processamento de Imagens, Otsu

ABSTRACT

FREITAS, Jovander da Silva. **Implementation of an architecture for FPGA image binarization.** 2012. 76f. Dissertation (Master) Departamento de Engenharia Elétrica e Computação, Universidade de São Paulo, São Carlos, 2012.

In many imaging applications it is desirable that images are converted to grayscale images to binary, ie with only two intensity levels. To accomplish this task separation between two levels is necessary to calculate a threshold value as determined from it which pixels will belong to a level generally the object of interest, and which belong to another level, or to the background image . Some applications require you to calculate this threshold value in a very short time in relation to image acquisition, especially when a very high brightness variation in the acquisition of an image. To meet this difficulty in the speed image processing applications, an alternative would be to develop an architecture dedicated to perform the calculation of the value of threshold and binarize the image acquired. This paper proposes the development of an architecture that performs these tasks by implementing reconfigurable circuits like FPGA. Making a comparison of results obtained with algorithms developed in Matlab, thus performing a validation of the proposed architecture. The developed architecture has reached the maximum frequency of 84.52 MHz, and the architecture can be operated in real-time system, using an image as a source of composite video or a regular camera.

Keywords: FPGA, Binarization, Image Processing, Otsu

LISTA DE ILUSTRAÇÕES

Figura 1 - Estrutura funcional de um processamento de imagem.....	27
Figura 2 - Imagem em tons de cinza (8bits).....	29
Figura 3 - Imagem binária.....	29
Figura 4 - (a) Imagem escura e seu respectivo histograma; (b) Imagem clara e seu respectivo histograma.....	31
Figura 5 - Histograma bimodal.....	33
Figura 6 - Imagem binarizada pelo método de Niblack.....	35
Figura 7(a) Imagem original alto contraste, (b) Imagem binarizada pelo método de Niblack, (c) Imagem original baixo contraste, (d) Imagem binarizada pelo método de Niblack.....	40
Figura 8(a) Imagem original alto contraste, (b) Imagem binarizada pelo método de Pun, (c) Imagem original baixo contraste, (d) Imagem binarizada pelo método de Pun...	41
Figura 9(a) Imagem original alto contraste, (b) Imagem binarizada pelo método de Otsu, (c) Imagem original baixo contraste, (d) Imagem binarizada pelo método de Otsu	42
Figura 10 - Arquitetura interna de um CPLD.....	43
Figura 11 - Arquitetura interna de um FPGA	44
Figura 12 - Módulos da arquitetura computacional de Otsu.....	47
Figura 13 - Arquitetura da implementação direta.....	48
Figura 14 - Arquitetura proposta para o cálculo da variância entre classes.....	49
Figura 15 - Diagrama de bloco do limiar.....	52
Figura 16 - Módulos da arquitetura computacional de Otsu.....	53
Figura 17- Módulo Estatístico de Otsu.....	54
Figura 18- Módulo de cálculo da arquitectura de Otsu.....	55

Figura 19- Esquemático da arquitetura implementada.....	57
Figura 20 - Diagrama de estado do método utilizando memória de Clock-2x.....	58
Figura 21 - Esquema da implementação da técnica Clock-2x no histograma de intensidade.....	59
Figura 22 - Esquema da implementação da técnica Clock-2x no histograma de área.....	60
Figura 23 - Esquema de implementação do histograma acumulado.....	61
Figura 24 - Esquema de implementação do módulo de área acumulada.....	61
Figura 25 - Esquemático das operações do módulo sigma quadrado.....	63
Figura 26 - Diagrama de operações simultâneas no módulo de sigma quadrado.....	64
Figura 27 – Módulo de Histograma.....	66
Figura 28 - Arquitetura do módulo de histograma acumulado implementado no Quartus.....	68
Figura 29 - Simulação do módulo de histograma acumulado.....	69
Figura 30 - Arquitetura do módulo sigamaquadrado.....	70
Figura 31 – Simulação da arquitetura do módulo sigamaquadrado.	71
Figura 32 – Valores de simulação do módulo sigamaquadrado.....	72
Figura 33 – (a) Imagem Original, (b) Histograma da imagem, (c) Imagem binarizada através do algoritmo da arquitetura, (d) Imagem binarizada utilizado o Otsu Matlab.....	72
Figura 34 - Componentes do módulo Otsu.....	73

LISTA DE SIGLAS

ASIC	Application Specific Integrated Circuit
CI	Circuito Integrado
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
IVC	International Verilog Conference
OCR	Optical Character Recognition
PLD	Programmable logic device
RAM	Random Access Memory
RGB	Red, Green, Blue
TTL	Transistor Transistor Logic
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

SUMÁRIO

DEDICATÓRIA	V
AGRADECIMENTOS	VII
EPÍGRAFE	IX
RESUMO	XI
ABSTRACT	XII
LISTA DE ILUSTRAÇÕES	XV
LISTA DE SIGLAS	XVII
SUMÁRIO	XIX
1. INTRODUÇÃO	21
1.1 OBJETIVOS	22
1.2 MOTIVAÇÃO	22
1.3 METODOLOGIA UTILIZADA	24
1.4 ORGANIZAÇÃO DA DISSERTAÇÃO	25
2. CONCEITOS BÁSICOS	27
2.1 IMAGENS EM NÍVEIS DE CINZA	27
2.2 IMAGENS BINÁRIAS	29
2.3 HISTOGRAMA	30
2.4 LIMIAÇÃO (THRESHOLDING)	31
2.5. MÉTODOS DE BINARIZAÇÃO	33
2.6 ALGORITMOS DE CÁLCULO DE LIMIAÇÃO	34
2.7 DISPOSITIVO RECONFIGURÁVEL	42
3.0 IMPLEMENTAÇÕES EM <i>HARDWARE</i>	47
3.1 ARQUITETURA DE TIAN, LAM, SRIKANTHAN (2003)	47
3.2 ARQUITETURA DE ASHARI E HORNSEY (2004)	51

3.3 ARQUITETURA DE JIANLAI et al. (2009)	53
4. IMPLEMENTAÇÃO E RESULTADOS	57
4.1 IMPLEMENTAÇÃO	57
4.2 RESULTADOS	73
4.3 CONSIDERAÇÕES FINAIS	74
5. CONCLUSÃO	75
REFERÊNCIAS	77
APÊNDICE A	79
APÊNDICE B	80

1. INTRODUÇÃO

Segundo Gonzalez e Woods (2010), uma imagem digital pode ser definida como uma função bidimensional, representada por $f(x,y)$, onde x e y são denominados coordenadas espaciais do plano, e a amplitude de f em qualquer coordenada de x e y é definida como intensidade da imagem neste ponto. Para definição de imagem digital é necessário que as intensidades de f sejam quantidades finitas e discretas. Uma imagem digital pode ser considerada como sendo uma matriz cujos índices das linhas e das colunas identificam um ponto na imagem, e o correspondente valor do elemento da matriz identifica a informação pictorial naquele ponto. Os elementos desta matriz são chamados de *picture elements*, mais conhecidos como pixels.

O Histograma é a base para numerosas técnicas de processamento no domínio espacial, de acordo com Gonzalez e Woods (2010). Ele representa uma distribuição estatística dos níveis de cinza de uma dada imagem digital em função da sua intensidade, ou seja, fornece a informação sobre a quantidade de pixels que uma determinada imagem possui com relação aos possíveis níveis de intensidade. Ao analisar um histograma, é possível analisar se uma imagem foi capturada em uma condição de maior ou menor luminosidade, analisar a relação entre partes da imagem, como fundo e objetos. Um problema encontrado com frequência em pesquisas que envolvem uso de histograma para processamento de imagens em tempo real é a variação de intensidade de luz sofrida pelo dispositivo de captura. Em alguns casos existe a necessidade de realizar a captura e já de imediato obter o histograma. O gráfico de um histograma retrata problemas encontrados no processamento de imagens em geral, como as condições de luminosidade sobre uma cena. A partir do histograma é possível realizar diversos processamentos no domínio espacial, como a binarização, utilizada na segmentação de imagens, e a equalização, para o tratamento visual da imagem.

Em várias aplicações de processamento de imagem é desejável converter uma imagem que está em escala de cinza para uma imagem de apenas dois níveis. Para extrair informações úteis a partir de uma imagem é preciso dividi-la em elementos distintos, por exemplo, objetos e fundo, para posterior análise da imagem. Esse processo de divisão é realizado através de técnicas de binarização, que consiste de cálculos matemáticos realizados através dos valores dos pixels da imagem a ser processada. Isso retrata um dos principais problemas encontrados em aplicativos de processamento de imagem. Os principais objetivos dos métodos apresentados se referem à eficiência em termos de desempenho da separação entre objeto e fundo e não abordam o termo velocidade.

1.1 OBJETIVO

O objetivo do trabalho está em desenvolver uma arquitetura dedicada para realizar o processo de binarização de imagens utilizando o método apresentado por Otsu (1979) para realizar o cálculo do valor de limiar. Toda arquitetura é desenvolvida em FPGA (*Field Programmable Gate Array*), utilizando a linguagem descrição de *hardware Verilog*.

1.2 MOTIVAÇÃO

Para algumas aplicações, especialmente aquelas que envolvem aplicações em tempo real, a velocidade é um requisito fundamental. Conforme Martins (2007), essas aplicações possuem requisitos específicos, tais como processamento de grande volume de dados e curtos intervalos de tempo para processar. Esse tipo de implementação em plataformas reconfiguráveis baseadas em dispositivos FPGA tem um volume cada vez maior na área de pesquisas em relação a outras tecnologias já consagradas, tais como a computação paralela, DSP (*Digital Signal Processor*) ou ASIC (*Application Specific Integrated Circuit*).

A justificativa do uso desta tecnologia deve-se a flexibilidade de adaptação e configuração, juntamente com o alto desempenho e seu baixo custo de aquisição.

Segundo Ashari e Hornsey (2004), uma implementação de uma técnica rápida e simples para binarização de imagens tem muitas aplicações, tais como processamento de imagens *on-chip* integrado com sensores CMOS (*Complementary Metal Oxide Semiconductor*). Aplicações em tempo real utilizando binarização também têm grandes influências em estudos de robótica, automóvel e monitoramento.

Outra aplicação de binarização em tempo real utilizando *hardware* dedicado é a implementação de técnicas de OCR (*Optical Character Recognition*) em dispositivos como *scanners*. Um *scanner* pode digitalizar e processar algumas páginas, para que esse processamento ocorra de forma mais eficiente, exige-se velocidade e processamento nesse sistema, sendo uma alternativa a arquitetura dedicada para realizar tal tarefa, que consiste em capturar a imagem, binarizar e realizar o reconhecimento dos caracteres da imagem.

Apesar de ser facilmente implementado em linguagens de programação de alto nível, tais como C, Python, Java, etc., há poucas referências que relatam qual a melhor forma de sua implementação e quais os melhores métodos para realizar tal tarefa em *hardware*, mais especificamente em dispositivos reconfiguráveis, como afirma Garcia (2000) e Shahbahrani (2008). Por isso há o interesse em se desenvolver um sistema que realize a binarização de imagens obtidas a partir de uma câmara de vídeo comum de forma eficiente e funcional.

A motivação em desenvolver uma arquitetura em FPGA, que realize a tarefa de calcular um valor de um limiar e converta uma imagem em apenas dois tons, se dá por meio de minimizar o problema em aplicações de processamento digital de imagens, quando há necessidade de calcular várias vezes o valor de limiar em um pequeno intervalo de tempo, por conta de uma transição grande de luminosidade no momento da captura da imagem. Sendo assim, neste trabalho é apresentada a implementação de um sistema desenvolvido em FPGA

da Altera para realizar os cálculos que envolvem a obtenção de imagens binárias a partir de uma imagem obtida por meio de uma câmera de vídeo.

1.3 METODOLOGIA UTILIZADA

A implementação e a análise dos métodos de limiarização, assim como simulações do funcionamento da arquitetura proposta, foram realizadas com a ferramenta computacional Matlab, da MathWorks versão R2008b.

Todo o desenvolvimento da arquitetura foi realizado na ferramenta computacional Quartus II 9.1sp2 Web Edition, da Altera. Foi utilizada a linguagem de descrição de *hardware* Verilog para a descrição dos algoritmos de binarização e realização da configuração do *hardware*.

A escolha da utilização de linguagens de descrição de *hardware*, ao contrário da implementação em esquemático, se dá pela facilidade da implementação em relação a outros métodos e a modificação mais fácil, tanto dos módulos, quanto da própria arquitetura. O uso da ferramenta computacional Quartus II se deve ao fato deste trabalho ser baseado no uso de dispositivos reconfiguráveis da própria Altera.

Toda a implementação física e os testes da arquitetura foram realizados no Kit DE2-70 de desenvolvimento baseados em FPGA da Altera, onde o mesmo já possui alguns componentes que são úteis para o desenvolvimento da arquitetura, tais como, conversor A/D, módulo de vídeo composto, etc.

Todas as simulações quanto a análise da arquitetura, incluindo cada módulo, para a validação do funcionamento da mesma, serão executadas no próprio Quartus II, e comparadas com as simulações realizadas com a ferramenta computacional Matlab.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

No capítulo 1 foi apresentada a introdução da dissertação e descritos os objetivos, a metodologia utilizada no desenvolvimento e a motivação para este trabalho.

No capítulo 2 é apresentado o conceito de binarização de imagens apresentando os métodos mais referenciados na literatura científica. Será apresentado o conceito de imagens em tons de cinza e imagens binárias. Esse capítulo também aborda o tema histograma e limiarização, dois assuntos de extrema importância para aplicações em processamento de imagens. Descreve a história e estudo dos algoritmos existentes na literatura para binarização de imagens tanto estaticamente como dinamicamente. Faz se uma breve introdução sobre os dispositivos reconfiguráveis.

No capítulo 3 é descrito as implementações em *hardware* para a binarização de imagens, tanto em arquiteturas reconfiguráveis como em circuitos dedicados, encontradas na literatura.

No capítulo 4 é apresentado a implementação realizada e os resultados obtidos.

No capítulo 5 é descrito a conclusão referente ao trabalho desenvolvido.

2. CONCEITOS BÁSICOS

Nesse capítulo será abordado o conceito de imagens em tons de cinza, o processo realizado para se transformar uma imagem em binária. Apresenta também a importância de um histograma e a limiarização no processamento de imagens e uma breve descrição dos dispositivos reconfiguráveis.

2.1 IMAGENS EM NÍVEIS DE CINZA

Segundo Facon (2005), o processamento digital de imagens pode ser classificado, quanto ao grau de abstração, em três níveis distintos: baixo, médio e alto. A classificação é devido à redução progressiva da quantidade de informações manipuladas, à medida que se passa por níveis crescentes de abstração.

No baixo nível, os dados de entrada do sistema de processamento de imagem são os pixels da imagem adquirida e os dados de saída são representações em valores numéricos associados aos pixels da entrada. No nível médio o que se difere do nível baixo é que a sua saída reproduz uma lista de características, também conhecido na literatura como vetor de características. Já no processamento de alto nível, é realizada uma interpretação a respeito do conteúdo da imagem, com base nas características extraídas no processamento anterior, conforme apresentado na Figura 1.

Figura 1 - Estrutura funcional de um processamento de imagem

Uma imagem digital pode ser representada pela $f(x,y)$, de acordo com Gonzalez e Woods (2010), representando os dois eixos da localização espacial de cada pixel. Além desta representação, em uma imagem em tons de cinza é necessário um terceiro eixo f (intensidades) como função das duas variáveis x e y . A intensidade sempre será representada por valores positivos e inteiros, que variam de acordo com a digitalização da imagem.

Segundo Gonzalez e Woods (2010), não existe uma restrição para o tamanho das dimensões de uma imagem, representada por uma matriz ($M \times N$), o único fato relevante é que essas dimensões devem ser valores inteiros e positivos. Já para L , que representa os níveis discretos de intensidade, é normalmente uma potência inteira de dois representado pela Equação 1.

$$L=2^k \quad (1)$$

No entanto, em algumas aplicações, devido ao armazenamento e processamento levado em consideração ao *hardware*, é de extrema importância realizar uma limitação destas representações, tanto quanto à dimensão da imagem ($M \times N$) quanto à quantidade de níveis de cinza. Por exemplo, numa imagem de 640 por 480 de resolução e com 256 de níveis de cinza, são necessários exatamente 2.457.600 bits de armazenamento. Normalmente não representamos uma imagem em bits. Encontramos na literatura apenas a resolução espacial, representada por números inteiros, e sua intensidade, sempre em bits. Por exemplo, a Figura 2 a seguir é uma imagem de 320 x 240 representada por 8 bits de intensidade de cinza para cada pixel, ou seja, em cada *pixel* da imagem poderá ter uma variação de 256 níveis de cinza.

Figura 2 - Imagem em tons de cinza (8bits)

2.2 IMAGENS BINÁRIAS

Uma imagem binária é representada por apenas um bit de intensidade, ou seja, apenas dois níveis de cinza, normalmente o branco e o preto. Utilizada na maioria dos algoritmos de análise de imagem, retirando informações relacionadas à geometria dos objetos, como área, perímetro ou quantificando estes elementos, segundo Coelho (2000).

Figura 3 - Imagem binária

Na Figura 3 é demonstrada uma imagem binária, onde seus *pixels* são representados apenas por dois tons, o branco e o preto. Normalmente, imagens binárias são

obtidas a partir de uma imagem com maior número de tons de cinza, por meio de um processamento conhecido por binarização ou limiarização.

2.3 HISTOGRAMA

O histograma de uma imagem digital de $M \times N$ *pixels* com intensidades no intervalo $[0, L - 1]$ é uma função discreta $h(r_k) = n_k$, onde r_k é o k -ésimo valor de intensidade e n_k é o número de *pixels* da imagem com intensidade r_k . É comum nas aplicações realizar a normalização do histograma, realizando uma divisão de cada componente descrito anteriormente pelo o número total de *pixels* da imagem que foi obtido o histograma. Desta forma, o histograma normalizado é dado pela Equação 2.

$$P_r(r_k) = \frac{n_k}{MN} . \quad (2)$$

O histograma, portanto, tem por objetivo demonstrar se os níveis de intensidade de uma imagem estão distribuídos adequadamente dentro dos possíveis níveis de cinza. O nível de intensidade varia de 0 (preto) ao 255 (branco), caso seja uma imagem de intensidade de tons de 8 bits.

Figura 4 - (a) Imagem escura e seu respectivo histograma; (b) Imagem clara e seu respectivo histograma
Fonte: Gonzalez e Woods, 2010.

Uma imagem dita escura, ou com baixo contraste possui um histograma cuja distribuição dos pixels se concentra mais à esquerda do gráfico (Figura 4 (a)), enquanto que em uma imagem dita clara, com alto contraste, a distribuição dos pixels está mais à direita (Figura 4 (b)). Em uma dita imagem ideal, há uma distribuição uniforme ao longo de todo histograma.

Ainda conforme Gonzalez e Woods (2010), o histograma é a base para várias técnicas de processamento de imagens no domínio espacial. A manipulação de histogramas pode ser utilizada para técnicas de realce de imagens, fornecer informações para técnicas de compressão e segmentação de imagens.

2.4 LIMIAZIZAÇÃO (*THRESHOLDING*)

Segundo Gazzoni (2006), na análise de objetos em imagem, é de grande importância destacar os objetos de interesse do resto da imagem, onde este resto pode ser

considerado o fundo da imagem. As técnicas para destacar os objetos de interesse são conhecidas na literatura atual e referenciadas como técnicas de segmentação de objetos. Uma das técnicas mais utilizadas é a limiarização (*thresholding*).

A técnica de limiarização consiste em segmentar uma imagem entre objeto e fundo utilizando um ponto de limiar, conhecido na literatura como ponto ou valor de limiar (*threshold*). A principal consideração no uso de um único nível de decisão capaz de classificar as regiões da imagem está baseada na condição de que as regiões (fundo e objeto) podem ser distinguidas usando-se apenas a informação dos níveis de cinza da imagem, conforme se refere Albuquerque (2008).

Para se chegar a um ponto de limiarização utilizando apenas os níveis de cinza contidos na imagem, é relatada na literatura atual a importância do histograma de intensidade da imagem. Neste caso, o histograma, por sua vez, retrata em alguns casos duas regiões bem características, conhecido na literatura como histograma bimodal, sendo uma das regiões contendo um pico elevado, que quantifica os pixels de tonalidade clara na imagem, que em alguns casos representa de alguma forma os objetos da imagem e a outra região representa neste caso o fundo da imagem, por ser observada no histograma como um pico mais baixo, que seria quantificação dos pixels mais escuro do fundo da imagem. Desta forma, a determinação do limiar é realizada achando-se um valor mínimo entre os picos existentes no histograma, como pode ser observado na Figura 5. Sendo essa forma conhecida como método do vale.

Figura 5 - Histograma bimodal.

Quando escolhemos um valor de limiar T , e iremos realizar uma limiarização, pode-se afirmar que, qualquer ponto (x,y) na imagem em que $f(x,y) > T$ é chamado de ponto do objeto ou pixel pertencente ao objeto. Caso contrário, o ponto é chamado ponto do fundo ou pixel pertencente ao fundo da imagem. Segundo Gonzalez e Woods (2010), uma imagem segmentada é dada pela Equação 3.

$$g(x,y) = \begin{cases} 1 & \text{se } f(x,y) > T \\ 0 & \text{caso contrário} \end{cases} \quad (3)$$

Embora não seja uma regra de se usar o valor 0 para os *pixels* do fundo e o valor 1 para os *pixels*, dois valores distintos podem ser utilizados. Quando se aplica um único valor de T para uma imagem inteira se é considerado uma limiarização global.

2.5. MÉTODOS DE BINARIZAÇÃO

Conforme citado por Mehmet (2004), há uma falta de medidas objetivas para avaliar o desempenho de algoritmos de binarização e dificuldades em realizar testes em ambiente de tarefas, ou seja, testar todos os algoritmos com o mesmo propósito. Com isso Mehmet (2004), que descreveu um estudo na tentativa de elaborar uma notação unificada de

uma variedade de algoritmos de binarização, categorizou os métodos limiarização em seis grupos, de acordo com que cada método explorava. As categorias apresentadas foram:

- Métodos baseados em histogramas;
- Métodos de agrupamento onde se agrupa duas partes o fundo e o objeto;
- Métodos baseados em entropia;
- Algoritmos de medida de similaridade entre os níveis de cinza;
- Métodos de uso espacial de ordem superior da distribuição de probabilidade entre os pixels
- Método local que tem por objetivo adaptar o valor limite de cada pixel da imagem com as características locais.

De acordo com essa avaliação é possível notar a variabilidade do desempenho dos algoritmos de caso para caso, de forma que um dos métodos apresentados é possível obter um resultado satisfatório para todas as imagens analisadas, ficando clara a inviabilidade de um método ser útil para todas as aplicações de processamento de imagem e variação de luminosidade.

2.6 ALGORITMOS DE CÁLCULO DE LIMIAR

Com a função de realizar a separação ou segmentação de um objeto com o fundo de uma imagem, Otsu (1979) desenvolveu um método de seleção automática do nível de limiar baseado nos valores dos pixels encontrados na imagem em escala de cinza a ser realizada a separação, sendo este método classificado nas categorias apresentadas por Mehmet (2004) em método baseado por histograma. Já o método apresentado por Niblack (1986) se enquadra na categoria de algoritmos de medida de similaridade entre os níveis de cinza. O método de Pun, descrito por Parker (1996) é um exemplo de métodos baseados em entropia. Abaixo descreveremos estes métodos existentes na literatura, em especial o método de Otsu.

2.6.1 Niblack

Um método conhecido na literatura é o de Niblack (1986), como um método de limiarização local de bom desempenho, que tem como objetivo construir uma superfície limite, baseada na média local e um desvio padrão de valores de cinza processados em uma vizinhança pequena ao redor de cada pixel da imagem. Esse método, entretanto, tende a produzir ruídos nas imagens, devido ao fato de trabalhar com vizinhanças.

Segundo Mello (2010), a imagem é dividida em janelas quadradas ($N \times N$) e cada janela é avaliada separadamente, onde calcula-se a média (m) e desvio padrão (d) de cada janela. O valor de limiar (th) é definido então por $th = m + bias * d$, $bias$ é o peso atribuído e não há definição de valor específico para esse $bias$.

Figura 6 - Imagem binarizada pelo método de Niblack
Fonte: Mello (2010)

Na Figura 6(a) é apresentada a imagem original e na Figura 6(b), a imagem binarizada, onde seu respectivo limiar foi calculado através do método de Niblack, tendo como parâmetro o valor da janela de 31 e $bias$ de -0.8. Observamos que por ser um método local o mesmo não distingue regiões uniformes, ressaltando pequenas variações que ocorrem nestas regiões, resultando em uma imagem ruidosa.

2.6.2 Pun

O método Pun apresentado por Parker (1996), de acordo com Olivete (2005), é baseado em entropia, que tem por teoria a medida de quantidade média de informação por símbolo de uma fonte. Por esta teoria, define a entropia do objeto e do fundo da imagem, e o limiar global é obtido através do cálculo de argumento máximo entre a entropia do objeto e do fundo. O algoritmo de Pun é apresentado por um ponto de corte t , calcula a entropia dos tons abaixo (H_b) e acima (H_w) de t , onde H_b é dado pela Equação 4 abaixo descrita e H_w pela Equação 5

$$H_b = - \sum_0^t p[i] \log(p[i]) \quad (4)$$

$$H_w = - \sum_{t+1}^{255} p[i] \log(p[i]) \quad (5)$$

O algoritmo deve procurar t que maximize H de forma apresentada na Equação 6.

$$H = H_w + H_b \quad (6)$$

2.6.3 Renyi

O método de Renyi, também baseado em entropia, é demonstrado por Mello (2010). Tem por objetivo otimizar uma função de critérios, onde usa duas distribuições de probabilidade, sendo uma o objeto e outra o fundo do objeto, probabilidades estas derivadas das distribuições de cinza da imagem original e inclui os métodos da soma da entropia máxima. Dado que $p_0, p_1, p_2, \dots, p_{255}$ são as probabilidades de níveis de cinza da imagem, duas probabilidades são derivadas delas $A1$ e $A2$, respectivamente uma para o objeto e outra para o fundo do objeto. A entropia de Renyi é da pela Equação 7 a seguir apresentada:

$$H_t^\alpha = \frac{1}{1-\alpha} \ln \sum_{k=1}^{255} (p_k^\alpha) \quad (7)$$

As entropias das classes são dadas pela Equação 8 e 9.

$$H_{A_1}^\alpha = \frac{1}{1-\alpha} \ln \sum_{k=0}^t \left(\frac{p_i}{p(A_1)} \right)^\alpha \quad (8)$$

$$H_{A_2}^\alpha = \frac{1}{1-\alpha} \ln \sum_{k=t+1}^{255} \left(\frac{p_i}{p(A_2)} \right)^\alpha \quad (9)$$

Por fim o valor de limiar é dado pela Equação 10.

$$t^*(\alpha) = \text{ArgMax} \{ H_{A_1}^\alpha(t) + H_{A_2}^\alpha(t) \} \quad (10)$$

Onde α é uma variável do tipo real e não sendo pré-definida.

2.6.4 Otsu

O método de Otsu é uma técnica popular de limiarização global automática, que pode ser utilizada em uma ampla gama de aplicações, tais como reconhecimento de veículos brasileiros, Freitas (2005), na extração de características de imagens carpais, Olivete (2005) e o pré-processamento de imagem de impressão digital, Emiroglu e Akhan (1997) entre outras.

Para se aplicar o método de Otsu (1979) em uma imagem digital é necessário que esta esteja em escala de cinza. O algoritmo é baseado em histograma, ou seja, a quantificação dos pixels em relação aos seus valores conforme demonstrado por Otsu (1979).

De acordo com Gonzalez e Woods (2010), o método de Otsu é considerado ótimo no sentido de maximizar a variância entre classes, onde essas classes representam uma atribuição de pixels para dois ou mais grupos. A operação inicial é calcular um valor que ofereça a melhor separação entre as classes, utilizando os valores de intensidade de pixels para realizar tal operação, esse valor seria o melhor limiar. Um relevante ponto do método de Otsu é o de se basear totalmente nos valores do cálculo do histograma de uma imagem, ou seja, independe de parâmetros adicionais.

O método consiste em primeiro realizar o cálculo do histograma de uma imagem, já explicado na sessão 2.3. Em segundo, suponha que temos um limiar $T(k) = k$, $0 < k < L-1$,

sendo L distintos níveis de intensidade de uma imagem digital de $M \times N$. Utilizamos o limiar para estabelecer duas classes C_1 e C_2 , normalmente fundo e objeto, em que C_1 possui todos os pixels com valores de intervalo de $[0, k]$ e C_2 o intervalo $[k+1, L-1]$. Utilizando esse limiar podemos atribuir à probabilidade $P_1(k)$, de um pixel ser atribuído a classe C_1 , que é dado pela somatória de todas as possibilidades do intervalo de C_1 , representada na Equação 11,

$$\omega_0 = \Pr(C_1) = \sum_{i=1}^k p_i = \omega(k) \quad (11)$$

Se analisarmos que a probabilidade de ocorrência da classe C_1 for 0, ou seja, estabelecermos $k=0$, podemos escrever a Equação 12, de forma que C_1 não terá pixel atribuído e a soma das classes devem C_1 e C_2 são necessariamente 1.

$$\omega_1 = \Pr(C_2) = \sum_{i=k+1}^L p_i = 1 - \omega(k) \quad (12)$$

Para que o método possa calcular o valor de limiar ótimo é necessário que obtenha o valor da intensidade média das classes C_1 e C_2 representados respectivamente pelas Equações 13 e 14.

$$\mu_0 = \sum_{i=1}^k i \Pr(i | C_1) = \sum_{i=1}^k i p_i / \omega_0 = \mu(k) / \omega(k) \quad (13)$$

$$\mu_1 = \sum_{i=k+1}^L i \Pr(i | C_2) = \sum_{i=k+1}^L i p_i / \omega_1 = \frac{\mu_T - \mu(k)}{1 - \omega(k)} \quad (14)$$

Sendo que na Equação 14 o μ_T é a intensidade média de toda a imagem, ou seja, a média global, que é dada pela somatória de todas as intensidades médias da classe C_1 e C_2 , representada pela Equação 15.

$$\mu_T = \mu(L) = \sum_{i=1}^L i p_i \quad (15)$$

Para consolidar as equações exposta até o momento, podemos realizar a substituição dos resultados anteriores e obtendo as seguintes Equações 16 e 17.

$$\omega_0 \mu_0 + \omega_1 \mu_1 = \mu_T \quad (16)$$

$$\omega_0 + \omega_1 = 1 \quad (17)$$

De acordo com Otsu (1979), a eficiência do limiar no nível de k , pode ser utilizada a medida de variância entre as classes e variância global, definida na Equação 18.

$$\eta = \frac{\sigma_B^2}{\sigma_T^2} \quad (18)$$

Sendo que σ_B^2 é a variância entre as classes C_1 e C_2 representada na equação 19.

$$\sigma_B^2 = \omega_0 (\mu_0 - \mu_T)^2 + \omega_1 (\mu_1 - \mu_T)^2 = \omega_0 \omega_1 (\mu_0 - \mu_1)^2 \quad (19)$$

E σ_T^2 representa a variância global, ou seja, a variância de intensidade de todos pixel da imagem, como indicado na equação 20.

$$\sigma_T^2 = \sum_{i=1}^L (i - \mu_T)^2 p_i \quad (20)$$

Desta forma, quanto mais distante as duas médias μ_0 e μ_1 estiverem uma da outra, maior será σ_B^2 , entendendo se assim que é uma medida de separação entre as classes. Como o σ_T^2 é um valor fixo que independe de k , entendendo desta forma podemos dizer que η é uma medida de separação entre as classes, concluído que a maximização de σ_B^2 é equivalente à maximização de η , desde que $\sigma_T^2 > 0$. Realizando essa operação determinamos o valor do limiar k^* , que maximiza a variância entre classes, representado na Equação 21.

$$\sigma_B^2(k^*) = \max_{1 \leq k < L} \sigma_B^2(k) \quad (21)$$

Onde, através das equações (11), (12), (13) e (14) é obtido a Equação 22 a partir da implementação da Equação 19.

$$\sigma_B^2 = \frac{[\mu_T \omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]} \quad (22)$$

A seguir são apresentadas algumas figuras binarizadas através dos métodos anteriormente descritos. Na figura 7(a) é apresentado uma imagem de alto contraste, sendo a figura 7(b) a mesma imagem binarizada através do método de Niblack (1986), onde o algoritmo foi implementado no Matlab, utilizado os parâmetros de janela 31 e de *bias* -0,8. Já na figura 7(c) é visualizado uma imagem de baixo contraste e na figura 7(d) a imagem binarizada também com o método de Niblack (1986), utilizando os mesmos parâmetros.

Figura 7(a) Imagem original alto contraste, (b) Imagem binarizada pelo método de Niblack, (c) Imagem original baixo contraste, (d) Imagem binarizada pelo método de Niblack

Na figura 8(a) é demonstrado uma figura original com alto contraste e logo em seguida a Figura 8(b) é a imagem binarizada através do método de Pun, onde pode se observar

na Figura 8(c) a imagem original com baixo contraste e na Figura 8(d) a imagem de baixo contraste binarizada através do método de Pun.

Figura 8(a) Imagem original alto contraste, (b) Imagem binarizada pelo método de Pun, (c) Imagem original baixo contraste, (d) Imagem binarizada pelo método de Pun

Na Figura 9(a) a imagem original de alto contraste e na Figura(9)b a imagem binarizada através do método de Otsu (1979), a Figura (9)c é uma imagem original de baixo contraste e a Figura (9) d é a imagem binarizada.

Figura 9(a) Imagem original alto contraste, (b) Imagem binarizada pelo método de Otsu, (c) Imagem original baixo contraste, (d) Imagem binarizada pelo método de Otsu

2.7 DISPOSITIVO RECONFIGURÁVEL

Inicialmente, os projetos de circuitos digitais implementados eram baseados em Circuitos Integrados (CI), mais comuns os TTL (*Transistor Transistor Logic*), onde o projetista possui uma dificuldade em realizar teste e simulações em projetos maiores, pois todos os CI teriam que ser interligadas em uma matriz de contato, também conhecido como *protoboard*, e isso normalmente causavam erros de interligação frequentes. De acordo com Pedrino (2008), os dispositivos lógicos programáveis, conhecidos como *Programmable Logic Device* (PLD) constituem uma ferramenta extremamente poderosa para projetistas de sistemas digitais nos dias atuais. Esses dispositivos são constituídos de vários conjuntos de portas lógicas e pode ser configurados através de um *software*, normalmente fornecido pelo seu fabricante. A arquitetura de um PLD possui uma malha de interconexão horizontal e faixas

verticais e em cada cruzamento é possível mapear quais cruzamentos não serão ligados, realizando assim um sistema digital, conforme Xilinx (2006).

Usualmente classificam-se os PLD em duas categorias:

- *Complex Programmable Logic Device (CPLD)*: possui blocos de macro células de PLD em um único dispositivo com o propósito geral de realizar uma interconexão entre essas células, conforme pode ser visualizado na Figura 10. Onde os caminhos lógicos simples podem ser implementados dentro de um único bloco, mais conforme a lógica fica mais complexa é necessário utilizar vários blocos que são interligados entre eles.

Figura 10 - Arquitetura interna de um CPLD

Fonte: <http://www.fpga-site.com>

- *Field Programmable Gate Array (FPGA)*: é uma estrutura regular de células lógicas, também conhecidas como módulos de células lógicas, possuem geralmente blocos de entrada e saída, conforme Figura 11 onde o projetista

tem controle total destas células, podendo realizar alterações em seu projeto sempre que desejar. Possui uma capacidade de portas lógicas significante ao se comparar com os CPLDs, as FPGAs são voláteis, o que implica na perda de sua configuração, no caso de ausência de energia elétrica, desta forma o FPGA deve ser programada toda vez que for iniciada, ou de alguma forma armazenar sua configuração em uma memória externa. Segundo Xilinx (2006) as FPGAs foram introduzidas em 1985 pela empresa Xilinx, onde provocou uma mudança nos projetos de circuitos digitais, sendo capaz de lidar praticamente com qualquer tarefa computacional de acordo com Pedrino (2008).

Figura 11 - Arquitetura interna de um FPGA

Fonte: <http://www.fpga-site.com>

Segundo Vivek (2002), a complexidade de projetos de *hardware* cresceu nas últimas décadas de forma exponencial, esse crescimento é alimentado por novos avanços em tecnologia de *design*, fabricação e o uso da linguagem de descrição de *hardware* para

modelar, simular e sintetizar. Na literatura atual as linguagens HDLs mais referenciadas são o VHDL e o Verilog.

VHDL é uma linguagem para descrever sistemas digitais eletrônicos, surgiu através do Governo dos Estados Unidos na necessidade de criar uma linguagem padrão para descrever as estruturas e as funções dos circuitos. A linguagem nasceu no início dos anos 80 e logo foi adotada como norma pelo Instituto de Engenheiros Elétricos e Eletrônicos (IEEE) nos Estados Unidos, conforme relatado por Ashenden (1990). VHDL foi projetado para atender as necessidades que um processo de *desing* de sistemas digitais precisa, tais como, a descrição da estrutura de interligação dos subprojetos e principalmente permitir que um projeto seja simulado antes que seja fabricado, dispensando assim a demora de montar um protótipo do *hardware*.

A linguagem Verilog criada pela Gateway Design Automation em 1985 e mais tarde desenvolvida pela Cadence Design Systems, está especificada no *standard* IEEE 1364 de 1995. De acordo com Vivek (2002) Verilog foi projetado para permitir descrições de projetos complexos e grandes, de modo que seja projetado de forma precisa e resumido, podendo facilitar nas descrições de arquiteturas de *design*, tais como *pipeline*, gestão de *cache*. Verilog HDL tem sido aplicado com sucesso em todas as grandes realizações no campo do design digital na última década, anualmente possui o *International Verilog Conference* (IVC) e é usado na grande maioria dos projetos da indústria eletrônica e informática.

3. IMPLEMENTAÇÕES EM *HARDWARE*

Há uma gama gigantesca de aplicações de processamento de imagens apresentada na literatura atual, mas quando se trata de processamentos de imagens em *hardware* esse assunto fica restrito e a poucos exemplos. Esse capítulo descreve algumas implementações relevantes para o presente trabalho, realizadas em dispositivos reconfiguráveis, que se aplicam à área de processamento de imagens.

3.1 ARQUITETURA DE TIAN, LAM, SRIKANTHAN (2003)

De acordo com Tian, Lam, Srikanthan (2003), sua arquitetura para o processamento do algoritmo proposto por Otsu (1979) é subdividida em três módulos de cálculos, conforme a Figura 12.

Figura 12 - Módulos da arquitetura computacional de Otsu

Fonte: Adaptado de Tian, Lam, Srikanthan (2003).

O módulo *Cumulative Histogram and Intensity Area Computation*, visualizado na Figura 9, tem por finalidade realizar os cálculos de histograma acumulado (CH) e os valores de área de intensidade acumulativa (CIA).

O histograma acumulado realiza os cálculos e armazena os valores de $\omega_o(t)$, para L níveis de cinza, sendo o $\omega_o(t)$ um valor de parâmetro para o próximo módulo que realiza o cálculo de variância entre as classes, propostas por Otsu (1979).

Já os valores da área de intensidade acumulativa (CIA) representam o valor de $\mu_t(t)$ para L níveis de cinza e armazena o valor de μ_T no topo de seus valores, sendo esses valores não normalizados e indispensáveis para o módulo seguinte.

Conforme Tian, Lam, Srikanthan (2003), o desenvolvimento de uma arquitetura que realize diretamente os cálculos do algoritmo proposto por Otsu (1979) é repleta de cálculos, conforme Figura 13, tais como multiplicação, divisão e raiz quadrada, que exigem um grande esforço computacional e requerem circuitos complexos.

Figura 13 – Arquitetura da implementação direta

Fonte: Tian, Lam, Srikanthan (2003).

O módulo *Between-class Variance Computation*, é composto de um LCU (*Logarithmic Conversion Unit*) binário, que converte a multiplicação e a divisão, propostos no método de Otsu (1979) e visualizado na Figura 12, em circuitos de ponto fixo simplificado.

Esse LCU é constituído de um multiplexador, duas LUTs (*Look-up table*) pequenas e uma unidade de cálculo simples. O multiplexador é utilizado para selecionar um dos três operandos em qualquer momento. Os outros recursos dentro da LCU consistem de deslocadores e somadores para manipular os conteúdos recuperados a partir da LUT para a conversão logarítmica. Pode se visualizada na Figura 14 a arquitetura proposta por Tian, Lam, Srikanthan (2003).

Figura 14 – Arquitetura proposta para o cálculo da variância entre classes

Fonte: Tian, Lam, Srikanthan (2003).

O limiar ótimo é calculado pelo módulo *Optimal Threshold Computation*, representado da Figura 12, onde o módulo realiza a comparação entre os valores fornecidos pelo módulo *Between-class Variance Computation*, fazendo-se a maximização desses valores.

Como a arquitetura trabalha com operações logarítmicas, há imprecisão do valor de Otsu e a necessidade de uma parametrização de valores para LCU. Pois, para realizar a parametrização, Tian, Lam, Srikanthan (2003) realizam a representação binária n bits de Z ($Z_{n-1}, Z_{n-2}, \dots, Z_0$) e Z_c é o primeiro bit de valor um para Z . O valor deste número é descrito na Equação 23, onde $0 \leq x < 1$. E o valor binário logarítmica de Z é expresso na Equação 24. Onde c é a característica do logaritmo e $\log_2(1+x)$ constitui a mantissa.

$$Z = 2^c + \sum_{i=0}^{c-1} 2^i z_i = 2^c \left(1 + \sum_{i=0}^{c-1} 2^{i-c} z_i \right) = 2^c (1+x) \quad (23)$$

$$\log_2 Z = c + \log_2(1+x) \quad (24)$$

De acordo com Tian, Lam, Srikanthan (2003), para se obter uma precisão razoável de conversão, um LUT pode ser usado para representar a mantissa, isto é $\log_2(1+x)$. Só que a profundidade da LUT terá uma complexidade de $O(2^n-1)$. À medida que é aumentado a

largura de dados de entrada o método torna-se dispendioso em relação à área utilizada e blocos de IO.

Para reduzir o tamanho da LUT, Tian, Lam, Srikanthan (2003) identifica os valores da mantissa com base nas combinações da palavra binária u bit formadas por $Z_{c-1} Z_{c-2} \dots Z_{c-u}$, onde u bits seguinte da posição c , são utilizados para tratar um LUT pequeno que armazena o L_{min} , ou seja o mínimo de mantissa de cada grupo.

Os m bits restante de Z ($Z_{c-u-1} Z_{c-u-2} \dots Z_{c-u-m}$) são utilizados para calcular um valor de precisão aproximada da mantissa. Estes bits são divididos em grupos multiplicadores $[m/2]$ conforme Equação 25.

$$f_k = Z_{c-u-(2k+1)}^Z Z_{c-u-(2k+2)}^Z \quad (25)$$

Os multiplicadores são calculados para cada combinação de $Z_{c-1} Z_{c-2} \dots Z_{c-u}$, que formam o conteúdo de outro LUT endereçado através de $Z_{c-1} Z_{c-2} \dots Z_{c-u}$. Os multiplicados são então deslocados de acordo com os grupos de multiplicadores. O algoritmo de aproximação completo pode expresso na Equação 26.

$$\log_2 Z = c + L_{min} + \sum_{k=0}^{[m/2]-1} ((d / 2^{2k}) f_k) \quad (26)$$

Tian, Lam, Srikanthan (2003) afirmam que a precisão da aproximação pode ser variada através dos factores de U e m , onde quanto maior o valores deste factores maior será o tamanho de LUT e mais unidades de cálculos, que no retorno oferece uma maior precisão e vice-versa. O método de conversão binária é empregado para calcular o valor $\sigma_B^2(t)$ como mostrado na Equação 27.

$$\log_2 \sigma_B^2(t) = \log_2 \frac{[\omega_0(t)\mu_T - \mu_t(t)]^2}{[1 - \omega_0(t)]\omega_0(t)} = 2 \log_2 [\omega_0(t)\mu_T - \mu_t(t)] - \log_2 \omega_0(t) \quad (27)$$

A função reversa do logaritmo não é necessária neste caso, o interesse é apenas em descobrir o logaritmo do limiar $\sigma_B^2(t)$ em seu valor máximo.

Na Tabela 1 é apresentada uma comparação da latência, área utilizada e blocos de IO, entre a arquitetura proposta e uma implementação sem a LCU.

Tabela 1 – Valores de comparação entre implementação direta e com LCU
Fonte: Adaptado por Tian, Lam, Srikanthan (2003)

	Direct implementation	Proposed implementation
Latency(ns)	362	132.0
Area (slices)	622	109
Number of IOBs	113	49

Conforme vimos o método de Otsu foi implementado utilizando uma unidade de conversão binária logarítmica para eliminar as divisões e multiplicações complexas representadas na Equação 22. Os autores descrevem que a implementação é cerca de 2,75 vezes mais rápida do que uma implementação utilizando o método tradicional de Otsu, ou seja, uma implementação que não utiliza conversão binária logarítmica para eliminar as divisões e multiplicações complexas. Afirma ainda que a implementação reduz por um fator de seis da área ocupada pela implementação direta do método de Otsu, Para a implementação foi utilizado Xilinx Virtex XCV800.

3.2 ARQUITETURA DE ASHARI E HORNSEY (2004)

Ashari e Hornsey (2004) implementou uma arquitetura em FPGA que realiza a binarização de imagens através de um método que utilizava de redes neurais com 2 pesos. Conforme descrito pelos próprios Ashari e Hornsey (2004), trata-se de um método computacionalmente simples e sem diminuir a eficiência do cálculo do limiar, o que torna um algoritmo adequado para implementação em *hardware*.

O *hardware* implementado possui três módulos, composto por uma unidade de controle de memória, unidade de atualização de peso e unidade de limiarização, conforme ilustrado na Figura 15.

Figura 15 - Diagrama de bloco do limiar
 Fonte: Ashari e Hornsey (2004).

A unidade de controle de memória gera sinais de tempo em tempo, para enviar e receber dados dos *pixels* da imagem que esta sendo processada. O módulo de atualização de peso calcula a média aritmética dos pesos e limiares. Cada pixel de entrada é lido a partir da unidade de controle de memória em comparação com os pesos e assim é realizada uma atualização dos pesos, realizando um fator de aprendizagem da rede neural. Quando uma imagem completa é processada, a unidade de limiarização determina o valor de limiar pela

média dos pesos. E desta forma realiza uma nova escrita na memória, sobrepondo os valores iniciais para que seja apresentada a imagem binária.

3.3 ARQUITETURA DE JIANLAI et al. (2009)

Jianlai et al.(2009), apresentam uma implementação do método de Otsu em tempo real, utilizando o *device* Cyclone II da Altera em um kit DE2-35 da Terasic. A arquitetura proposta é bem próxima da descrita por Tian, Lam, Srikanthan (2003). Jianlai et al. (2009) afirmam que a implementação em FPGA leva vantagem pela capacidade de computação paralela oferecida neste tipo de *hardware*. Para o desenvolvimento Jianlai et al. (2009) separam o procedimento de cálculo de limiar do Otsu em três módulos, conforme representado na Figura 16.

Figura 16 - Módulos da arquitetura computacional de Otsu

Fonte: Jianlai et al. (2009).

No módulo *Statistics* são realizadas quatro funções: o histograma, histograma acumulado, cálculo de área de intensidade e área intensidade acumulada. Todas as quatro funções baseadas no histograma.

De acordo com Jianlai et al. (2009), o histograma é a peça chave para o módulo *Statistics* e para realizar o processo ele utilizou três espaços de memória para armazenar o resultado temporário do histograma, e quando se termina de montar o histograma o mesmo é copiado em uma nova memória obtendo assim histograma final. Através deste processo, elimina-se o conflito entre ler, escrever e acumular os valores para cada pixel antes de lidar com um novo valor de pixel, afirmando assim que o processo irá reduzir a velocidade do histograma.

Para implementação do módulo é utilizado uma memória RAM de porta dupla, conhecida como *dual port*, e é utilizado o valor do pixel como endereço de leitura para uma porta, realizando assim o incremento deste valor antes de ser escrito de novo na mesma posição através da segunda porta.

Para realizar o controle do histograma Jianlai et al. (2009) definiram uma máquina de estado simples onde o sinal válido é o positivo, assim ativando os módulos de histograma secundários, respetivos $3n^{th}$, $3n+1^{th}$, $3n+2^{th}$ em ciclos de clock. Definindo assim que cada módulo de histograma *Sub* terá três ciclos de clock para recolher informação de um pixel, realizando desta forma os passos de leitura, acumulação e escrita, e lidando com um pixel em cada ciclo de clock, conforme ilustrado na Figura 17.

Figura 17- Módulo Estatístico de Otsu
Fonte: Jianlai et al. (2009).

Na arquitetura proposta Jianlai et al. (2009) a resolução da imagem é de 320x240 e cada pixel é tratado com dez bits, sendo que cada memória que compõe o histograma é composta com 32 bits em cada posição, cada módulo de memória RAM possuem 1024 posições. O módulo que implementa o cálculo da variância entre as classes utiliza dois divisores e três multiplicadores para conseguir realizar o cálculo de σ_B^2 , utilizando o IP core da Altera *lpm_divide* e *lpm_mult* realizando o cálculo em cinco estágio de clock utilizando a técnica de *pipeline*, como pode ser observado na Figura 18.

Figura 18- Módulo de cálculo da arquitetura de Otsu
Fonte: Jianlai et al.(2009).

Já o módulo *Optimal Threshold Computation* é responsável por realizar o processo de comparação para escolher o σ_B^2 máximo e informar o limiar ótimo através do índice correspondente.

Para verificar a arquitetura proposta, Jianlai et al. (2009) utilizaram o *device* da Altera Cyclone ® II FPGA 2C35, a resolução da imagem foi definida em 320x240, a frequência de quadros foi de 45 quadros/segundo. E conclui que uma arquitetura do método de Otsu implementada em um dispositivo reconfigurável de forma eficiente pode ser empregada em vários métodos de processamento de imagens em tempo real.

Ao analisar as arquiteturas descritas anteriormente, onde todas implementam métodos de binarização de imagem em *hardware* é possível concluir que a arquitetura de Tian, Lam, Srikanthan (2003) e Jianlai et al. (2009), tem particularidades em comum, como a implementação do método de Otsu, sendo que a arquitetura de Tian, Lam, Srikanthan (2003), utiliza uma unidade LCU, para amenizar a complexidades dos cálculos existentes no método de Otsu (1979). Já a arquitetura desenvolvida por Jianlai et al.(2009), tem como característica a forma de obtenção do histograma, utilizando-se de três módulos de memórias para tratar os conflitos na hora de leitura e escrita, talvez por não terem conhecimento de técnicas tais como

Clock-2x, utilizada no presente trabalho. Esta mesma calcula o valor do limiar utilizando a Equação (19), com os valores não normalizados

A arquitetura de Ashari e Hornsey (2004) se difere das demais descritas, por utilizar rede neural que tem sua aprendizagem através da comparação dos valores de pixel da imagem com os pesos, realizando desta forma a atualização dos mesmos, que são utilizados para cálculo do valor de limiar.

4. IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo é descrito a implementação realizada em FPGA e os resultados obtidos através da mesma e comparados com resultados obtidos no Matlab.

4.1 IMPLEMENTAÇÃO

A arquitetura proposta na dissertação é um módulo que realiza os cálculos proposto por Otsu (1979), onde esse módulo tem por finalidade realizar todas as etapas de cálculos, tais como histograma, histograma acumulado, área acumulativa e os valores de σ_B^2 , como pode ser observado na Figura 19 onde é apresentado um esquemático da arquitetura implementada.

Figura 19- Esquemático da arquitetura implementada

O módulo de histogramas tem a finalidade de realizar o cálculo de dois histogramas, sendo um o histograma de intensidade não normalizado, explicado na sessão 2.3, e o outro o histograma de área não normalizado. Para a implementação foi utilizado uma técnica conhecida como Clock-2x, onde o clock de entrada da memória é dobrado para que as operações de leitura e escrita na memória sejam realizadas em único ciclo de clock do sistema, evitando assim o conflito de acesso dos endereços de memória entre o ciclo de leitura

e o ciclo de escrita. Essa técnica pode ser visualizada na Figura 20, onde é possível verificar um diagrama de estados de leitura e escrita.

Figura 20 – Diagrama de estado do método utilizando memória de Clock-2x
Fonte: MATHA 2012

Na primeira parte do ciclo de clock (CLK) é feita a leitura da memória. O valor lido (DOA) é incrementado por meio de um contador e o resultado é armazenado no mesmo endereço (ADDRRA) na parte final do ciclo de clock (CLK). O próprio sinal de clock (CLK) é utilizado para controlar a leitura e a escrita da memória. Esta técnica é utilizada nesta implementação, pois tem um menor consumo de células lógicas, de acordo com Matha (2012), não exige a implementação de componentes sequenciais extras, tais como contadores, nem força o atraso na entrada de dados (DELAY), permitindo o processamento em tempo real. Um esquemático desta implementação pode ser visualizado na Figura 21.

Figura 21 – Esquema da implementação da técnica Clock-2x no histograma de intensidade

Com o emprego da técnica de Clock-2x, o processo de geração do histograma utiliza o valor do nível de cinza do pixel atual (INPUT VALUE) tanto como endereço de leitura (rdaddress_a) como de escrita (waddress), por meio do segundo multiplexador. O dado (qa) é lido durante o ciclo inicial do clock, onde é incrementado por meio de um somador e escrito na memória (data) no ciclo final do clock, por meio do primeiro multiplexador. Para a descarga do histograma, um contador externo é utilizado para a geração dos endereços para a leitura dos dados (READ_ADDR), por meio do segundo multiplexador, e consequentemente realiza a limpeza da memória, armazenando do valor 0 em todas as posições, por meio do primeiro multiplexador. O controle destas operações é realizado por meio de multiplexador e do sinal de controle (SEL).

A mesma arquitetura é utilizada para gerar o histograma de área, sendo que o histograma de área é obtido através da soma dos valores dos pixels, conforme é demonstrado na Figura 22.

Figura 22 – Esquema da implementação da técnica Clock-2x no histograma de área

O módulo de histograma acumulado tem um papel fundamental para realização do cálculo de σ_B^2 proposto por Otsu (1979). O histograma acumulado obtém a acumulação dos valores do histograma de intensidade, desta forma, a arquitetura desenvolvida necessita aguardar a realização dos cálculos de todos os itens do histograma, para posteriormente realizar o cálculo do histograma acumulado, que é expresso na Equação 28. Onde é realizada uma somatória de todos, sendo n_i os itens do histograma de intensidade.

$$Histograma_Acumulativo(k) = \sum_{i=0}^k n_i \quad (28)$$

Os valores do histograma acumulado são referentes ao $\omega(k)$ que representam a classe C_1 detalhada na sessão 2.6.4. Sendo que o último valor acumulado representa a totalidade de pixel da imagem, ou seja, $M \times N$, conforme visualizado na Figura 23. Obtendo assim dois parâmetros indispensável para o cálculo de σ_B^2 utilizado no módulo sigma quadrado.

Figura 23 – Esquema de implementação do histograma acumulado.

O módulo de área acumulada se refere à somatória das posições 0 a k do histograma de área, acumulando o valor das posições anteriores à posição atual, conforme expresso pela Equação 29.

$$area_acumulada(k) = \sum_{i=0}^k in_i \quad (29)$$

Sendo que o topo desta somatória se dá o valor de μ_T , que é um dos valores requeridos para o módulo de sigma quadrado e cada item do módulo de área acumulada resulta no $\mu(k)$. Essa estrutura é visualizada na Figura 24.

Figura 24 – Esquema de implementação do módulo de área acumulada

Após obter os valores referentes ao histograma acumulado de intensidade e área acumulada possível realizar o cálculo de $\sigma_B^2(k)$, apresentado na sessão 2.6.4 conforme Equação 30.

$$\sigma_B^2 = \frac{[\mu_T \omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]} \quad (30)$$

Observando a Equação 30, deparamos com cálculos complexo, tais como potenciação, multiplicações e divisões, que de certa forma exige que o *hardware* trabalhe com representações de números com alta quantidade de bits e a necessidade de calcular para cada σ_B^2 um valor de $\omega(k)$ e $\mu(k)$ o que se torna uma implementação complexa, pela necessidade de trabalhar com ponto fixo. Para amenizar essa representação e o ponto fixo, reescreveu-se a equação através das Equações (13), (14) e (15), chegando se à Equação 31, simplificado de certa forma a implementação em *hardware*.

$$MN^2 \sigma_B^2(k) = \frac{(\mu_T \omega(k) - \mu(k)MN)}{(\omega(k)MN - \omega(k)^2)} (\mu_T \omega(k) - \mu(k)MN) \quad (31)$$

Onde MN representa o tamanho da imagem em pixels, e é obtido através do último elemento do histograma acumulado, que por sua vez fornece os valores de $\omega(k)$. Já os valores de μ_T e $\mu(k)$ são oriundos do módulo de área acumulada, onde o μ_T é obtido através do topo do vetor que armazena os valores de $\mu(k)$.

O módulo de sigma quadrado tem a função de realizar os cálculos matemáticos apresentados na Equação 31, sendo que é preciso realizar uma comparação entre os valores, para que o possa ser encontrado o valor máximo e assim o limiar ótimo, maximizando as classe C_1 e C_2 de acordo com Otsu (1979).

A arquitetura do módulo de sigma quadrado é realizada através de quatro multiplicações simultâneas, realizadas através de técnicas de *pipeline*, recurso muito utilizado nas FPGA, sendo que posteriormente são realizadas duas subtrações utilizando a mesma técnica, ilustrado na Figura 25. Esta estrutura de multiplicação/subtração é encontrada em FPGAs que possuem módulos de DSP, como por exemplo, os *devices* da família Stratix da fabricante Altera.

Figura 25 – Esquemático das operações do módulo sigma quadrado

Conforme visualizado na Figura 25, é necessário após os cálculos de multiplicação e subtração a implementação de um registrador (DFF), para realizar o atraso da informação, realizando assim o sincronismo com a operação de divisão. Finalizando os cálculos é realizada uma multiplicação. Na Figura 25 é demonstrada a quantidade significativa de bits necessário em cada operação, totalizando no final a quantidade de 64 bits para os valores de σ_B^2 . Já a realização da comparação é realizada item a item, onde um registrador é incrementado quando a comparação entre o valor calculado de σ_B^2 é maior que o valor de σ_B^2 .

armazenado em outro registrador. Desta forma é possível obter o maior σ_B^2 em um registrador, e o registrador incrementado armazena o limiar considerado ótimo através do método apresentado por Otsu (1979).

Na Figura 26, é apresentado o diagrama de passos de como é realizado as operações matemática até chegar o valor de σ_B^2 .

Figura 26 – Diagrama de operações simultâneas no módulo de sigma quadrado

Ao observar a Figura 26 pode se notar que após os valores de entrada chegar ao módulo é necessários cinco pulsos de clock para obter o primeiro valor de σ_B^2 , os demais são obtidos com um único pulso de clock, pois a arquitetura estará alimentada com valores contínuo e realizando operações em *pipeline*.

A primeira fase da implementação do projeto foi o módulo que realiza o cálculo do histograma, para isso foi necessário utilizar o *MegaWizard Plug-In Manager*, utilitário encontrado no *software* Quartus II. Através do *MegaWizard Plug-In Manager* foi projetado uma memória RAM (*Random Access Memory*) de 19 bits para os valores e 8 bits para endereçamento.

A memória projetada utiliza três portas, sendo que uma das portas é utilizada para a leitura do valor atual do histograma (`rdaddress_a`), a segunda realiza a atualização dos valores (`waddress`), e uma terceira porta (`rdaddress_b`) é utilizada tanto para descarregar o histograma da memória (`qb`) como para zerar (0) seu conteúdo para gerar um novo histograma. A implementação da memória descrita na figura 21 é apresentada na figura 27.

Além da memória RAM de três portas, a implementação do histograma utiliza o valor de nível de cinza do pixel atual (`endereço_pixel`), visualizado na Figura 27, que demonstra a arquitetura implementada no Quartus II. Esse valor de pixel é utilizado tanto como endereço de leitura (`rdaddress_a`), durante o processo de geração do histograma, como de escrita (`waddress`), durante o processo de leitura do histograma e reinicialização do mesmo, por meio do segundo multiplexador. O dado (`qa`) é lido durante o ciclo inicial do clock e é incrementado por meio de um somador e escrito na memória (`data`) no ciclo final do clock, por meio do primeiro multiplexador. Para a descarga do histograma, um contador externo é utilizado para a geração dos endereços para a leitura dos dados (`READ_ADDR`), por meio do segundo multiplexador, e conseqüentemente realiza a limpeza da memória (armazenamento do valor 0), por meio do primeiro multiplexador. O controle destas operações é realizado por meio de multiplexador e do sinal `SEL`.

Figura 27 – Módulo de histograma

Na implementação do módulo de histograma acumulado, foram utilizados dois contadores internos de 8 bits, sendo o primeiro utilizado para gerar os endereços de gravação na memória quando o sinal (sel) de está ativo, já o segundo é utilizado para gerar os endereços quando se deseja zerar os valores da memória, servido assim para limpar a memória gravando valores zeros, para que possa ser calculado um novo histograma acumulado.

O módulo de histograma acumulado tem um papel fundamental na arquitetura, pois os valores calculados através de sua arquitetura, conforme visualizada na Figura 28, são fundamentais para obtenção do valor de limiar.

Figura 28 – Arquitetura do módulo de histograma acumulado implementado no Quartus

Na figura 29, pode se observar uma simulação realizada no *software* Quartus II do módulo histograma acumulado, sendo que os valores de entrada (valor_hist) foram gerados a partir de uma sequência de números representados em hexadecimal. Já através dos valores de saída (dados_saida) é possível observar a acumulação dos valores quando o sinal (sel) está ativo, assim que é desativado toda saída é zero.

Figura 29 – Simulação do módulo de histograma acumulado

O módulo de área acumulada tem grande semelhança de implementação do módulo do histograma acumulado, sendo a principal diferença que os valores de entrada são oriundos do histograma de área, realizando assim uma acumulação destes valores, que tem como representação máxima 27 bits.

Para implementar o módulo do sigmaquadrado, foi utilizado nas operações matemáticas o *MegaWizard* para gerar as unidades de operação, sendo todos os cálculos realizados com números inteiros, eliminando assim trabalhar com ponto fixo. A arquitetura do módulo de sigmaquadrado é composta por cinco unidades lpm_mul, que realiza as multiplicações apresentadas na Equação 31, duas lpm_add_sub, realizando subtração, uma

unidade Imp_divide e um Imp_compare que é utilizado para comparar os valores de σ_B^2 , obtendo assim o maior valor. Essa arquitetura pode ser observada na Figura 30.

Figura 30 – Arquitetura do módulo σ_B^2

A Figura 31 mostra uma simulação realizada no módulo de sigma quadrado tendo como valores de entrada o sinal (ω_0), oriundo do módulo de histograma acumulado explicado anteriormente, o sinal (m_iT) e o sinal (m_i0) que são calculados no módulo de área acumulada. A simulação é realizada com valores gerados no *software* Matlab, realizando assim uma comparação dos valores obtidos na arquitetura com os valores obtidos no Matlab.

Figura 31 – Simulação da arquitetura do módulo sigamaquadrado.

Os valores apresentados na simulação são referentes às posições 124 a 131 dos valores gerados no Matlab, sendo que o limiar é dado na posição 126. Desta forma a simulação apresentou o limiar com valor 3, pois foram calculados e comparados 8 valores de σ_B^2 , sendo o de maior valor presente na terceira posição. Conforme pode ser visualizado na Figura 32.

1	Indice	122	123	124	125	126	127	128	129	130	131	132	133
2	histograma	212	213	246	217	241	211	216	208	241	216	249	226
3	hexadecimal	D4	D5	F6	D9	F1	D3	D8	D0	F1	D8	F9	E2
4													
5	histograma_area	25652	25986	30258	26908	30125	26586	27432	26624	31089	28080	32619	29832
6	hexadecimal	6434	6582	7632	691C	75AD	67DA	6B28	6800	7971	6DB0	7F6B	7488
7													
8	histograma_acc	213436	213649	213895	214112	214353	214564	214780	214988	215229	215445	215694	215920
9	hexadecimal	341BC	34291	34387	34460	34551	34624	346FC	347CC	348BD	34995	34A8E	34B70
10													
11	histograma_area_acc	14357123	14383109	14413367	14440275	14470400	14496986	14524418	14551042	14582131	14610211	14642830	14672662
12	hexadecimal	DB1283	DB7805	DBEE37	DC5753	DCCD00	DD34DA	DDA002	DE0802	DE8173	DEEF23	DF6E8E	DFE316
13													
14	sigma_quadrado	2,6772E+14	2,67767E+14	2,67807E+14	2,67828E+14	2,67836E+14	2,67829E+14	2,67808E+14	2,67775E+14	2,67772E+14	2,67658E+14	2,6757E+14	2,67475E+14
15													

Figura 32 – Valores de simulação do módulo sigamaquadrado.

Os valores demonstrado na Figura 32 são referentes ao processamento da imagem da Figura 33(a), onde se utilizou essa imagem para realizar as simulações da arquitetura proposta, fazendo se uso dos valores para realizar a comparação. A Figura 33(b) permite visualizar os valores do histograma de intensidade referente à imagem.

Figura 33 – (a) Imagem Original, (b) Histograma da imagem, (c) Imagem binarizada através do algoritmo da arquitetura, (d) Imagem binarizada utilizado o Otsu Matlab.

Já a Figura 33(d) é possível visualizar a imagem binarizada através do algoritmo implementado na arquitetura, sendo que o limiar foi de 126. Na Figura 33(d), a imagem binaria foi obtida através do método de Otsu, disponível no *toolbox* do Matlab, sendo o limiar apresentado de 126, o que demonstra a validade da arquitetura de desenvolvida.

A arquitetura implementada, descrita na figura 19, pode ser visualizada na Figura 34, onde é possível visualizar o módulo de histograma, responsável pelos histograma de intensidade e histograma de área, em sequência esta os módulos de cálculos de histograma acumulado e área acumulada, e finalizando o módulo responsável pelo cálculo dos σ_B^2 e suas respectivas comparações.

Figura 34 - Componentes do módulo Otsu.

4.2 RESULTADOS

A arquitetura implementada no Quartus II apresentou um consumo de 944 células lógicas, sendo que 708 destas são registradores. A implementação também apresentou um consumo de 720896 bits de memória. Apesar de ser esperado um consumo de 23552 bits de memória, o consumo excessivo se deu pelo fato da FPGA possuir tamanhos fixos de memória que nem sempre correspondem ao tamanho solicitado pela implementação. Através da compilação pelo Quartus, onde se utilizou os parâmetros *default* do mesmo, constatou-se que a frequência máxima de operação da arquitetura é de 84,52 MHz. Esta frequência máxima está acima da frequência de clock de um *codec* para aquisição de imagem obtidas por câmeras de vídeo composto, como o que existe no Kit DE2-70 da Terasic, que opera a 27MHz. Esta frequência máxima também é superior à frequência de atualização de pixels de um vídeo composto, que é de 13,5 MHz.

Para obtenção do limiar e a binarização de cada imagem, a arquitetura gera, durante o processo de apresentação da imagem, tanto o histograma como o histograma de

área. Finalizado o histograma, são necessários 256 pulsos de clock para a geração do histograma acumulado, que é obtido através da leitura acumulativa do histograma. Concorrentemente ao histograma acumulado, também é gerado o histograma de área acumulada, pelo mesmo procedimento a partir do histograma de área. Para realizar a descarga das informações, tanto do histograma acumulado quanto do histograma de área acumulada, são necessários mais 256 pulsos de clock, totalizando, assim, um total de 512 pulsos de clock para que as informações cheguem ao módulo de sigma quadrado.

No módulo de sigma quadrado foram necessários 5 pulsos de clock para que fosse realizado o cálculo de σ_B^2 e a comparação para a identificação do valor de limiar, totalizando assim 517 pulsos de clock. Desta forma, o módulo de Otsu implementado calcula o valor de limiar em 19,15 μ s, considerando uma frequência de clock de 27MHz, comum em *codecs* de vídeo. Como o tempo de retraço horizontal do vídeo composto de 63,5 μ s, em 60Hz, concluído assim que o valor de limiar pode ser utilizado na próxima imagem a ser apresentada, considerando que não haja uma grande variação na imagem adquirida, pois o tempo de processamento da arquitetura é cerca de 1/3 do tempo de uma linha vídeo composto.

4.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados às implementações em *software* e em *hardware* do presente trabalho, de acordo com os objetivos propostos e com os resultados obtidos e comparados. Em toda a implementação se deu o cuidado de desenvolver uma arquitetura que possa ser aplicada a trabalhos que necessitam de uma binarização de imagens.

5. CONCLUSÃO

Após o levantamento das referências bibliográficas referentes aos métodos de binarização, foi verificada a existência de uma grande variedade de algoritmos e que há uma falta de medidas objetivas para avaliar o desempenho dos mesmos e dificuldades em realizar testes comparativos de forma eficiente, e que não existe um único método que seja ideal para todas as possíveis aplicações. Contudo, verificamos que o método proposto por Otsu (1979) possui a vantagem de não necessitar de parâmetros externos e que utiliza somente as informações das distribuições de níveis de intensidade dos pixels para obter o melhor valor de limiar, sendo também um método global.

Dada estas características do método apresentado por Otsu (1979), verificou-se que as poucas implementações em *hardware* de métodos para binarização são baseadas neste algoritmo. Contudo, as implementações encontradas na literatura apresentam limitações, como o uso de LUTs para o cálculo do limiar para evitar o uso de operações matemáticas complexas (Tian, Lam, Srikanthan (2003)), quanto no tamanho da imagem a ser processada (Jianlai et al.(2009)). Por isso, o desenvolvimento baseado no método proposto por Otsu, (1979), para que haja o cálculo do limiar de forma rápida entre uma aquisição de imagem e outra, a implementação de uma arquitetura específica, ou seja, dedicada para isso é de grande importância. Uma forma encontrada foi à utilização de dispositivos tipo FPGA, que permite realizar a configuração do dispositivo e reconfiguração, facilitando de certa forma a implementação. Já os cálculos do método de Otsu (1979) se torna complexo e de grande representação numérica, caso não seja tratado isso na implementação, como foi descrito neste trabalho.

Também foram apresentados exemplos de aplicações reais que podem fazer uso de tal arquitetura, já que se trata de uma arquitetura composta por módulo, onde o principal

valor de entrada é os valores do pixel da imagem, tornando assim independente de valores adicionais para realizar o cálculo do limiar.

Para futuros trabalhos, pode se destacar a realização de simulações em tempo real, e o desenvolvimento de uma máquina de estado para realizar o controle de sincronismo dos módulos que compõe a arquitetura desenvolvida. Outro ponto de grande interesse é desenvolver uma arquitetura com método recursivos de Otsu, para realizar a comparação de elementos tais como: velocidade, representações numéricas e uma arquitetura que implemente o método de Otsu para multilevel e demais itens de interesse.

REFERÊNCIAS

ALBUQUERQUE, M. P.; ESQUEF, I. A., *Image Segmentation using nonextensive relative entropy*, IEEE LATIN AMERICA TRANSACTIONS, VOL. 6, NO. 5, SEPTEMBER 2008

ASHARI, E., HORNSEY, R. *FPGA Implementation of Real-Time Adaptive Image Thresholding*, Photonics North 2004: Optical Components and Devices. Edited by Armitage, John C.; Fafard, Simon; Lessard, Roger A.; Lampropoulos, George A. Proceedings of the SPIE, Volume 5578, pp. 410-419, 2004.

ASHENDEN P. J., *"The VHDL Cookbook*. Dept. Computer Science University of Adelaide South Australia, 1990

COELHO, S. F. P., *Localização e leitura automática de caracteres alfanuméricos – uma aplicação na identificação de veículos*, Universidade Federal do Rio Grande do Sul, mestrado em Engenharia elétrica, 2000.

EMIROGLU, I. AND AKHAN, M.B. *Pré-Processing of Fingerprint Image*. European Conference on Security and Detection, Conference Publication N° 437, IEE, 1997, pp:147-151.

FACON, J. *Processamento e Análise de Imagens*, Pontifícia Universidade Católica do Paraná, Mestrado em Informática, 2005.

FREITAS, J. S.; REHAN, S. *Reconhecimento de placas de veículos brasileiros*. 2005. 100f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade de Franca, Franca.

GARCIA, E., *"Implementing a Histogram for Image Processing Applications,"* Xilinx, Xcell Magazine, (38), pp. 40-46, 2000.

GAZZONI, J. C.; MARCON C, L.; FRANCISCON, M. A.; MORAES, F. B., *Limiarização e binarização na análise de objetos em imagens digitais*, Synergismus scyentifica UTFPR, Pato Branco, 2006

GONZALEZ, R. C. E WOODS, R. E., *Processamento Digital de Imagens 3ª edição*. Pearson, 2010.

JIANLAI, W., CHUNLING, Y., MIN, Z., CHANGHUI, W., *Implementation of Otsu's Thresholding Process Based on FPGA*, Industrial Electronics and Applications, 2009. ICIEA 2009. 4th IEEE Conference

MARTINS, S. A. M. F., *Ferramenta de apoio ao desenvolvimento de sistemas de processamento de imagem em tempo real implementados em plataformas reconfiguráveis*, Departamento de Engenharia Electrotécnica e de Computadores, Porto, 2007

MATHA, M. H., *Integration of reduction operators in a compiler for FPGAs*, Dept. of Electrical and Computer Engineering, Colorado State University, http://www.radar.colostate.edu/ece/current_students/grad_exams/matha.pdf - Acesso em: 09 Jul. 2012.

MEHMET, S.; SANKUR, B. L., *Survey over image thresholding techniques and quantitative performance evaluation.* Journal of Electronic Imaging 13(1), 146–165 January 2004.

MELLO, C. A. B., *Arquivos de Aulas,* Disponível em: <<http://www.cin.ufpe.br/~cabm/visao/>>. Acesso em: 20 set. 2010.

NIBLACK, W., *An introduction digital image processing,* E. Cliffs, Ed. New York: Prentice Hall, 1986, pp. 115-116.

OLIVETE, J. C.; QUEIROZ, A. C., RODRIGUES, E. L. L., *Comparação entre métodos de limiarização aplicados na extração de características de imagens carpais.* X Congresso Brasileiro de Física Média, Bahia 2005, Brasil.

OTSU, N., *A Threshold selection method from gray level histograms.* IEEE Trans. System Man and Cybernet, 9, pp. 62-66, 1979.

PARKER, J. R., *Algorithms for Image Processing and Computer Vision,* John Wiley & Sons, New York, 1996

PEDRINO, E. C., *Arquitetura pipeline reconfigurável através de instruções geradas por programação genética para processamento morfológico de imagens digitais utilizando FPGAs,* tese de doutorado, EESC/USP, 2008.

SHAHBAHRAMI, A., HUR, J. Y., JUURLINK, B. H. H., WONG, S., “*FPGA Implementation of Parallel Histogram Computation,*” In Proc. 2nd HiPEAC Workshop on Reconfigurable Computing, pp. 63-72, Gothenburg, Sweden, January 2008.

TIAN H., LAM, S. K., SRIKANTHAN, T., *Implementing Otsu’s Thresholding Process Approximation Unit Using Area-Time Efficient Logarithmic,* Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium.

VIVEK, S., *The Complete Verilog Book,* Kluwer Academic Publishers, 2002

XILINX, *Programmable Logic Design Quick Start Hand Book,* second edition, 2006

APÊNDICE A – Publicação gerada durante o período de mestrado

FREITAS, J. S. ; LUPPE, M. . Cálculo de Histograma em FPGA para Processamento de Imagens em Tempo Real. In: VII Workshop de Visão Computacional, 2011, Curitiba-PR. Anais do VII Workshop de Visão Computacional, 2011. p. 103-106.

APÊNDICE B – Códigos da arquitetura em Verilog

```

module Otsu(
    input          iCLK,
    input          clk2x,
    input          reset,
    input          VS,
    input          VGA_Read,
    //input        [10:0] VGA_X,
    //input        [10:0] VGA_Y,
    input          [7:0] Valor_Pixel,
    output         [7:0] Level
);

reg sel;

wire [18:0] Fio0;
wire [26:0] Fio1;
wire [26:0] Fio2;
wire [18:0] Fio3;
wire [26:0] Fio4;

histo_control Histograma (
    .iCLK(iCLK),
    .clk2x(clk2x),
    .VS(VS),
    .VGA_Read(VGA_Read),
    //.VGA_X(VGA_X),
    //.VGA_Y(VGA_Y),
    .Valor_Pixel(Valor_Pixel),

    .val_hist(Fio0),
    .val_hist_area(Fio1)
);

histograma_acc Histograma_Acumulado (
    .iCLK(iCLK),
    .clk2x(clk2x),
    .sel(sel),
    .valor_hist(Fio0),
    .dados_saida(Fio3)
);

histograma_area_acc Histograma_Area_Acumulado(
    .iCLK(iCLK),
    .clk2x(clk2x),
    .sel(sel),
    .valor_area(Fio1),
    .miT(Fio4),
    .dados_saida(Fio2)
);

sigmaquadrado SigmaQuadrado(
    .iCLK(iCLK),
    .clk2x(clk2x),
    .ativa(sel),
    .omega0(Fio3),
    .miT(Fio4),

```

```

        .mi0(Fio2),
        .Level(Level)

);

endmodule

module histo_control(
    input          iCLK,
    input          clk2x,
    input          VS,
    input          VGA_Read,
    //input [10:0] VGA_X,
    //input [10:0] VGA_Y,
    input [7:0] Valor_Pixel,
    output [18:0] val_hist, // Saída dos valores da memória do histograma
    output [26:0] val_hist_area // Saída dos valores da memória do
    histograma de area
);

reg [7:0]    Address;
reg         enable;
reg [7:0]    counter_add;
reg [7:0]    contador;
reg         en;

wire [18:0] Histo_data;
wire [18:0] qd;

parameter    tons_cinza = 255;

always @ (posedge iCLK)
    begin
        if (VS == 1'b1) // Quando estiver apresentando a imagem
            begin
                contador = 0;
                counter_add = 0;
            end
        else // Quando estiver fazendo o sincronismo
            begin
                if (contador < tons_cinza)
                    begin
                        contador = contador + 1;
                        en = 1'b1; // enable ativado
                    end
                else
                    en = 1'b0; //enable desativado
                    if (contador <= tons_cinza)
                        counter_add = contador;
            end
        Address <= counter_add;
        enable <= en;
    end

hist_area Histograma_area (
    .iCLK(iCLK),

```

```

        .sel(VS & VGA_Read),
        .clk2x(clk2x),
        .endereco_pixel(Valor_Pixel),
        .endereco_contador(Address),
        .dados_saida(val_hist_area)
    );

histograma Histograma(
    .iCLK(iCLK),
    .sel(VS & VGA_Read),
    .clk2x(clk2x),
    .endereco_pixel(Valor_Pixel),
    .endereco_contador(Address),
    .dados_saida(val_hist)
);

endmodule

module histograma_area_acc (
    input          iCLK,
    input          sel,
    input          clk2x,
    input [26:0]   valor_area, // valor do histograma de area
    //input [7:0]   endereco_contador, // Endereços gerados pelo
    contador para zerar ou extrair dados
    output reg [26:0] miT,
    output [26:0]   dados_saida
);

reg [7:0]   contador;
reg [7:0]   contador_dois;
reg [26:0]  dados;
reg [7:0]   wraddress;

initial
    contador = 8'd1;

initial
    contador_dois = 8'd0;

initial
    dados = valor_area;

always @ (posedge iCLK)
begin

    //Controle do 'data'
    if (sel == 1'b1) // VS = 1 Ciclo de Escrita (contagem de distâncias)
    ou VS = 0 -- Ciclo de Leitura (zeragem da memória)
    begin
        if (contador==8'd255)
        begin
            contador<=1;
            miT <= dados;
        end
    else
        dados <= dados + valor_area;
    end
end

```

```

        wraddress = contador;
        contador<=contador+1;
    end
else // Aqui ele le os dados da memoria Periodo de sincronismo
    begin
        if (contador_dois==8'd255)
            contador_dois<=0;
        else
            dados <= 0;
            wraddress = contador_dois;
            contador_dois<=contador_dois+1;
        end
    end
end

mem_2port_27bits Memoria_Hist_Area_ACC_27_bits(
    .data(dados), // Para gerar o histograma de omega0 na tela
    .inclock(clk2x),
    .outclock(iCLK),
    .rdaddress(wraddress),
    .wraddress(wraddress),
    .wren(iCLK),
    .q(dados_saida)
);

/*

mem_3port ram_3port (

    .clock(clk2x), // Clock da Memoria, ou seja, dobrado
    .data(dados), // Dados de entrada na memoria
    .wraddress(wraddress), // Ler a entrada de endereço para a memória.
    .wren(0), // 0 para testar com arquivo .mif
    //.wren(iCLK), // iCLK para real time

    //Porta A da memoria (Ativo em VS=1)
    .rdaddress_a(wraddress), //Usado apenas na contagem das
    distâncias
    .rden_a(~(iCLK) &(sel)), //Ativo no Ciclo de Escrita (VS =
1)
    .qa(dados_saida), //A ser incrementado

    //Porta B da memoria (Ativo em VS=0)
    .rdaddress_b(wraddress), // Usado apenas na contagem
    das distâncias
    .rden_b(iCLK &(~(sel))), // Ativo no Ciclo de Leitura
    (VS = 0)--was not clk
    .qb(0),
);
*/

endmodule

module sigmaquadrado (
    input iCLK,
    input ativa,
    input clk2x,
    input [18:0] omega0,
    input [26:0] miT,

```

```

        input      [26:0]      mi0,
        output reg  [7:0] Level

);

wire  [45:0] Fio0; // Estagio01
wire  [45:0] Fio1; // Estagio02
wire  [38:0] Fio2; // Estagio03
wire  [38:0] Fio3; // Estagio04
wire  [45:0] Fio4; // Estagio05
wire  [38:0] Fio5; // Estagio06
wire  [45:0] Fio6; // Estagio07
wire  [45:0] Fio7; // DFF
wire  [92:0] Fio8; // sigmaquadrado
wire  Fio9; // comparador

parameter MN = 307200; // Tamanho da imagem 640x480

// estagio01(x) = (hist_area_acc(256)* histograma_acc(x));
multiplicacao_27_19 Multiplicacao01(
    // .clock(iCLK),
    .dataa(miT), // 27 bits
    .datab(omega0), // 19 bits
    .result(Fio0) // 46 bits
);

multiplicacao_27_19 Multiplicacao02(
    // .clock(iCLK),
    .dataa(mi0), // 27 bits
    .datab(MN), // 19 bits
    .result(Fio1) // 46 bits
);

multiplicacao_19_19 Multiplicacao03(
    // .clock(iCLK),
    .dataa(omega0), // 19 bits
    .datab(MN), // 19 bits
    .result(Fio2) // 38 bits
);

multiplicacao_19_19 Multiplicacao04(
    // .clock(iCLK),
    .dataa(omega0), // 19 bits
    .datab(omega0), // 19 bits
    .result(Fio3) // 38 bits
);

subtracao_46_46 Subtracao01(
    // .clock(iCLK),
    .dataa(Fio0), // 46 bits
    .datab(Fio1), // 46 bits
    .result(Fio4) // 46 bits
);

subtracao_38_38 Subtracao02(
    // .clock(iCLK),
    .dataa(Fio2), // 38 bits
    .datab(Fio3), // 38 bits
    .result(Fio5) // 38 bits
);

```

```

Registrador DFF(
    .clock(iCLK),
    .data(Fio5), // 46 bits
    .q(Fio7) // 46 bits
);

Divisao Divisao01(
    //.clock(iCLK),
    .numer(Fio4), // 38 bits
    .denom(Fio5), // 38 bits
    .quotient(Fio6) // 38 bits
);

multiplicacao_45_45 Multiplicacao05(
    //.clock(iCLK),
    .dataa(Fio7),
    .datab(Fio6),
    .result(Fio8) // 92 bits
);

reg [92:0] maiorsigma;

initial
Level = 8'd0;
initial
maiorsigma = 92'd0; //Sigma2 começa valendo 0 para ser comparado com o
sigma1

always @ (posedge iCLK)
begin
    if (ativa==1'b1)
        if (Fio9==1'b1) //Se fio9 for igual 1 então é sigmacalculado é maior que
o sigma do registrador
            Level <= Level + 1;
            maiorsigma <= Fio8;

end

comparador compara01(
    .dataa(Fio8), // Primeiro sigma calculado e assim por diante
    .datab(maiorsigma), // Segundo sigma calculado assim por diante
    .agb(Fio9) // Saida 1 quando dataa é maior que datab, caso 0 dataa é
menor
);

endmodule

```