LEONARDO DE AMARAL VIDAL

ARQUITETURA EM PIPELINE PARA O ALGORITMO DE CANNY EM UMA PLATAFORMA VHDL/FPGA

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre do Programa de Pós-Graduação em Informática, apresentada ao Departamento de Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.

Orientador: Prof. Dr. Eduardo Todt

CURITIBA

2014

LEONARDO DE AMARAL VIDAL

ARQUITETURA EM PIPELINE PARA O ALGORITMO DE CANNY EM UMA PLATAFORMA VHDL/FPGA

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre do Programa de Pós-Graduação em Informática, apresentada ao Departamento de Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.

Orientador: Prof. Dr. Eduardo Todt

CURITIBA

2014

V648a Vidal, Leonardo de Amaral
Arquitetura em pipeline para o algoritmo de Canny em uma plataforma VHDL/FPGA / Leonardo de Amaral Vidal. – Curitiba, 2014.
147f. : il. color. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Exatas, Programa de Pós-graduação em Informática, 2014.

Orientador: Eduardo Todt .
Bibliografia: p. 45-50.

1. Processamento de imagens - Análise. 2. Circuitos integrados. 3. Programação paralela (Computação). I. Universidade Federal do Paraná. II.Todt, Eduardo. III. Título.

CDD: 621.3994

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Leonardo de Amaral Vidal, avaliamos o trabalho intitulado, *"Arquitetura em pipeline para o algoritmo de canny em uma plataforma VHDL/FPGA"*, cuja defesa foi realizada no dia 16 de setembro de 2014, às 09:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 16 de setembro de 2014.

Prof. Dr. Eduardo Todt
DINF/UFPR – Orientador

Prof. Dr. Alessandro Zimmer
Dept. Eng. Elétrica/UFPR – Membro Externo

Prof. Dr. Bruno Müller Junior
DINF/UFPR – Membro Externo

Prof. Dr. Roberto André Hexsel
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

UFPR

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

ATA DA DEFESA DE DISSERTAÇÃO DE MESTRADO EM INFORMÁTICA ALUNO: LEONARDO DE AMARAL VIDAL

No dia 16 de setembro do ano de dois mil e quatorze, às 09:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná, foi realizada a sessão pública da defesa de Dissertação de Mestrado em Informática do aluno Leonardo de Amaral Vidal. Estavam presentes, além do candidato, os Membros da Comissão Examinadora composta pelos Professores Eduardo Todt(Orientador), Alessandro Zimmer, Bruno Müller Junior e Roberto André Hexsel. Após a apresentação do trabalho do candidato, intitulado *"Arquitetura em pipeline para o algoritmo de canny em uma plataforma VHDL/FPGA"*, o mesmo foi arguido pela Comissão. A seguir, a Comissão reuniu-se em local reservado e decidiu, por unanimidade, pela aprovação do candidato, condicionada as alterações sugeridas pela mesma. O resultado foi então comunicado ao candidato e aos presentes na sessão pública. A seguir, o Presidente declarou encerrada a sessão da qual eu, Jucélia Miecznikowski, Secretária da Pós-graduação em Informática, lavrei a presente Ata, que depois de aprovada será assinada por mim, pelo Presidente, e pelos demais membros da Comissão.

Prof. Dr. Eduardo Todt
DINF/UFPR – Orientador

Prof. Dr. Alessandro Zimmer
Dept. Eng. Elétrica/UFPR – Membro Externo

Prof. Dr. Bruno Müller Junior
DINF/UFPR – Membro Externo

Prof. Dr. Roberto André Hexsel
DINF/UFPR – Membro Interno

Jucélia Miecznikowski
Secretária da PPGInf

DEDICATÓRIA

Dedico este trabalho a minha mãe Neusa de Amaral Vidal e ao meu pai Odilon Vidal pelo apoio e suporte incondicionais que me deram.

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Eduardo Todt por acreditar em mim mais do que eu mesmo. Agradeço aos colegas do grupo VRI (Visão, Robótica e Imagem) Victor Hugo Schulz que foi praticamente um coorientador para mim, Diego Addan Gonçalves, Felipe Bombardelli, Jeferson Fernando Guardezi, Luiz Augusto Volpi do Nascimento, Nicole Janny Salomons, Renan Traba e Rodrigo Gonçalves de Oliveira, que, em vários momentos, me ajudaram nesta minha jornada.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Os algoritmos de detecção de bordas necessitam de um poder muito alto de processamento, devido à quantidade de convoluções, problema agravado no caso de aplicações que exigem processamento de video em tempo real, como em robótica móvel. Uma maneira de melhorar o desempenho é implementar o algoritmo diretamente em hardware. Esta dissertação descreve um projeto de uma implementação do algoritmo de detecção de bordas Canny, realizada com a linguagem de descrição VHDL e com a linguagem de programação C++, em uma plataforma híbrida. A suavização, o cálculo do gradiente, a supressão de não máximos e o *threshold* duplo estão implementados em um computador de mesa do tipo PC (*Personal Computer*) e a segunda etapa da histerese está implementada em um FPGA (*Field Programmable Gate Array*), modelo Virtex 6, da Xilinx.
A arquitetura da parte implementada no FPGA é em *pipeline* e paralela.

Palavras-chave: Canny; FPGA; *Hardware* Reconfigurável; VHDL; Processamento de Imagens; Detecção de Bordas; Arquitetura Paralela; Arquitetura Híbrida; *pipeline*.

ABSTRACT

The edge detection algorithms require a very high power processing due the number of convolutions, an issue in real-time video applications like mobile robotics. One way to improve performance is to implement the algorithm directly in hardware. This paper describes and demonstrates the results of an implementation of the edge detection Canny algorithm performed with VHDL and the C++ programming language in a hybrid platform i.e.; Noise reduction, gradient intensity finding, non-maxima supression and double thresholding are implemented on a Desktop Personal Computer and the second part of hysteresis is implemented in a Xilinx Virtex 6 FPGA (Field Programmable Gate Array). The architecture designed on FPGA is a pipeline and parallel type.

Keywords: Canny; FPGA; Reconfigurable Hardware; VHDL; Image Processing; Edge Detection; Parallel Architecture; Hybrid architecture; pipeline.

CAPÍTULO 1

INTRODUÇÃO

O processamento de imagens é uma área da computação muito ampla e possui um leque enorme de aplicações no cotidiano. A detecção de placas de carro e a manipulação de imagens de exames médicos são exemplos desta aplicação.
Um componente importante desta área é a detecção de bordas, cujo o objetivo é extrair o contorno dos objetos contidos numa imagem. Muito utilizado em algoritmos de extração de características e segmentação de objetos, com seu uso é possível obter uma redução drástica na quantidade de dados a serem processados. Ou seja, pode ser filtrado o que for irrelevante, mantendo as propriedades relevantes da imagem para esses algoritmos.
Os algoritmos de detecção de bordas necessitam de um poder muito alto de processamento, devido a quantidade de convoluções, problema agravado no caso de aplicações que exigem processamento de video em tempo real, como em robótica móvel.
Esta dissertação descreve uma implementação híbrida do algoritmo de Canny, que é um dos detectores de bordas mais robustos da atualidade [41]. Uma parte do algoritmo é implementado em um computador de mesa do tipo PC *Personal Computer*) e a outra parte é implementada em uma plataforma FPGA (*Field Programmable Gate Array*). A parte contida no FPGA foi desenvolvida na linguagem de descrição VHDL e possui uma arquitetura que combina os formatos de *pipeline* e massivamente paralelo.

1.1 Motivação

A motivação é explorar as possibilidades de processamento paralelo oferecidas por plataformas baseadas em FPGAs e o paralelismo intrínseco identificado no algoritmo de Canny, para possibilitar o uso em aplicações com requisitos de tempo real, como robótica móvel, por exemplo.

1.2 Objetivo

O objetivo desta dissertação é projetar e analisar uma implementação do algoritmo de Canny com a combinação das plataformas PC e FPGA (Xilinx Virtex-6), com o uso das linguagens VHDL e C++, com o intuito de verificar as possibilidades de implementação de arquiteturas paralelas e em *pipeline* em FPGAs, tomando como caso particular o detector de bordas Canny. O meio escolhido para entrada e saída das imagens entre a plataforma FPGA e o computador hospedeiro foi o PCI Express, devido à sua

velocidade de comunicação elevada (5.0 Gbps) e ao fato de a plataforma FPGA poder ser embutida em um computador pessoal.

1.3 Metodologia

A metodologia consiste em primeiro dividir o algoritmo Canny em módulos conforme o tipo de processamento dos dados em cada etapa, que são o filtro Gaussiano, o gradiente, a supressão não máximos e a histerese. Cada módulo é implementado e testado em VHDL, se tiver com alvo ser residente em FPGA, ou então testado com Linux e OpenCV, se residente no PC. Todo sistema é testado gerando-se, a partir de imagens padrão, os resultados de bordas com Canny em OpenCV e com o sistema desenvolvido. Os resultados de cada solução são comparados por meio de uma métrica, que avalia de forma quantitativa a coincidência ou não das bordas produzidas.

1.4 Organização do texto

Além deste capítulo de introdução, este trabalho apresenta os seguintes capítulos e anexos:

- Capítulo 2 - Trabalhos Relacionados: são descritos o algoritmo de Canny, as ferramentas e as tecnologias utilizadas para o desenvolvimento deste trabalho;
- Capítulo 3 - Projeto: o sistema proposto é descrito, sendo suas entidades componentes, conexões e controle apresentados em detalhes;
- Capítulo 4 - Resultados e discussões: são apresentadas as métricas e os resultados das simulações realizadas;
- Capítulo 5 - Conclusão: são apresentados os pareceres finais sobre o trabalho executado
- Anexo A - Relatórios de Compilação: são apresentados os relatórios de compilação dos binários para o FPGA;
- Anexo B - Trechos de Código-Fonte: são apresentados trechos de código-fonte utilizados e desenvolvidos na linguagem C para este trabalho;
- Anexo C - *Scripts* para Automação de Processos: são apresentados os *scripts* desenvolvidos para automação de processos para o trabalho exposto;
- Anexo D - Tabelas com resultados por imagem: são apresentados as tabelas com os resultados dos testes para cada imagem da base adotada;

- Anexo E - Trechos de Código-Fonte na linguagem de descrição VHDL: são apresentados trechos de código-fonte projetados na linguagem de descrição VHDL.

CAPÍTULO 2

TRABALHOS RELACIONADOS

Este capítulo tem como objetivo dar uma visão do algoritmo de Canny (Seção 2.1), implementações relevantes publicadas e as tecnologias utilizadas para suportar o desenvolvimento da arquitetura proposta.

- Seção 2.2 - Arquivo de imagem PGM: descreve o formato de imagem que será utilizado neste trabalho;
- Seção 2.3 - VHDL: proporciona um breve relato sobre a linguagem empregada para implementar a arquitetura proposta;
- Seção 2.4 - FPGA: faz um breve descrição sobre o *hardware* a ser empregado;
- Seção 2.5 - Implementações do algoritmo de Canny em plataformas FPGA: descreve implementações correlatas a este trabalho;
- Seção 2.6 - Barramento PCI-Express: descreve o meio de comunicação que será utilizado neste trabalho;
- Seção 2.7 - PIO: descreve o método de transferência de dados adotado neste trabalho;
- Seção 2.8 - Plataforma de desenvolvimento Xilinx Virtex-6: descreve a plataforma de desenvolvimento que será utilizada.

2.1 Algoritmo de Canny

O algoritmo de Canny, criado por John Canny em 1983 [8] e publicado em 1986, é um detector de bordas otimizado. Para ser considerado um detector de bordas otimizado, o algoritmo deve possuir os seguintes critérios [9]:

1. Boa detecção: A probabilidade ao falhar na identificação de pontos que pertençam, de fato, a uma borda devem ser baixas, assim como deve ser baixa a probabilidade de identificar como borda os pontos que não pertencem à borda alguma (provavelmente ruídos na imagem). Uma vez que essas duas probabilidades são funções monotonicamente decrescentes do sinal de saída em relação a taxa de ruído, este critério corresponde a maximizar a magnitude do sinal em relação a taxa de ruido [9];

2. Boa localização: Os pontos marcados como parte de alguma borda pelo operador devem ser os pontos exatos, ou mais proximos possíveis, de uma borda real [9];

3. Apenas uma resposta para uma borda: Fator implicitamente definido em uma boa detecção, uma vez que dois resultados próximos correspondem à mesma borda, apenas um deles é considerado um borda falsa. Entretanto, a forma matemática do primeiro critério não atende ao requisito de múltiplas respostas e deve ser feito de forma explícita [9].

O algoritmo de Canny é composto de diversos estágios, executados em sequência sobre uma imagem de entrada, descritos a seguir:

1. Redução de ruídos: neste estágio é realizada uma convolução gaussiana, na imagem de entrada, para remoção de ruídos de alta frequência em relação aos sinais da imagem;

2. Gradiente e Magnitude: após ser feita a fase de redução de ruídos, é necessário encontrar o gradiente e a magnitude em cada pixel da imagem. Podem ser utilizados operadores tais como Roberts' Cross [48], Prewitt [46], Scharr [49] ou Sobel [51], para obter as derivadas primeira na direção horizontal (G_y) e vertical (G_x). Com as derivadas na direção horizontal e vertical, podemos obter o ângulo (θ) e a magnitude de intensidade (M) através das equações 2.1 e 2.2.

$$\boxed{\theta = \arctan\left(\frac{G_y}{G_x}\right)} \tag{2.1}$$

$$\boxed{M = \sqrt{G_x^2 + G_y^2}} \tag{2.2}$$

   O ângulo da direção da borda é arredondado para um dos ângulos que representam as direções vertical, horizontal e as diagonais do plano bidimensional. Esse arrendondamento foi proposto por Jain et al. [16, 31] e não consta na descrição original [9].

3. Supressão de não máximos: Dado o ângulo (θ) e a magnitude de intensidade (M) do gradiente, é analisado se o ponto pertence a um máximo local. Caso não pertença, é suprimido. O algoritmo de supressão de não máximos funciona da seguinte forma:

   (a) Para cada posição (x, y), veja as duas direções perpendiculares ao ângulo de orientação θ na posição (x, y);

   (b) Marque o pixel inicial (x, y) por P_i e os dois pixels na direção perpendicular à θ na posição (x, y) por P_1 e P_2;

   (c) Se $M(P_1) > M(P_i)$ ou $M(P_2) > M(P_i)$, então suprima P_i atribuindo $NMS(P_i) = 0$. Senão, $NMS(P_i) = M(P_i)$. Onde NMS é a imagem convoluída pela supressão de não máximos.

4. Histerese: Etapa final do algoritmo de Canny, é uma forma de segmentação (*thresholding*) da imagem, que possui dois limiares, H e L, tal que $H > L$. A primeira fase da histerese, funciona da seguinte maneira para cada posição (x, y) da imagem convoluída da supressão de não máximos (NMS):

   (a) Se $NMS(x, y) > H$, então marcar o pixel (x, y) como borda (B);

   (b) Se $NMS(x, y) < L$, então marcar o pixel (x, y) como não-borda (N);

   (c) Se $NMS(x, y) > L$ e $NMS(x, y) < H$, então marcar o pixel (x, y) como candidato a borda (C);

   A segunda fase do processo de histerese consiste em, transformar em borda todos os candidatos que possuem uma conexão com algum pixel que é borda. Portanto, existe uma sequência de pixels candidatos que formam um caminho entre um pixel candidato I e um pixel J, que é borda. Os pixels candidatos que não se enquadram nessa regra, tornam-se não-borda.

2.2 Arquivo de imagem PGM

O formato de arquivo de imagem utilizado é o PGM (*portable graymap format*) que foi escolhido por não possuir compressão de dados, o que facilita a sua manipulação. Este formato foi criado por Jef Poskanzer[23] em 1988. A motivação dessa criação era possibilitar enviar imagens como um texto ASCII puro através de e-mails. A especificação completa pode ser encontrada em [45]. Na implementação descrita nesta monografia foi utilizado o formato PGM com descrição textual (codificação *ASCII*) de forma simplificada, mas que atende a especificação original, da seguinte maneira:

1. Identificação do formato do arquivo que são os caracteres "P2"(PGM com codificação *ASCII*);
2. Caractere de nova linha;
3. Largura da imagem em quantidade de pixels (não há limite para a largura);
4. Caractere de nova linha;
5. Comprimento (altura) da imagem em quantidade de pixels (não há limite para o comprimento);
6. Caractere de nova linha;

7. Valor máximo inteiro da escala de cinza na imagem, que pode variar de 1 até 65535 de acordo com a especificação. Mas, foi usado o valor 255 neste trabalho;

8. Caractere de nova linha;

9. O pixel da primeira linha e da primeira coluna seguido do caractere de nova linha, depois vem o pixel da primeira linha com a segunda coluna também seguido do caractere de nova linha e assim sucessivamente até o último pixel da imagem. Todos os pixels podem variar de 0 (preto) até o valor máximo inteiro da escala de cinza informado no item 7 (branco).

Uma observação quanto a existência de uma grande quantidade de caracteres de nova linha é que este serve para separação dos dados contidos na imagem e, na especificação original, podem ser utilizados também os caracteres de tabulação e/ou de espaço em branco.

2.3 VHDL

VHDL (*VHSIC Hardware Description Language*) é uma linguagem de descrição de sistemas eletrônicos digitais, desenvolvida sob a tutela do Departamento de Defesa dos Estados Unidos para o projeto *Very High Speed Integrated Circuits* (VHSIC) na década de 1980.
A linguagem foi criada com o intuito de documentar circuitos integrados de uso específico. Entretanto, a linguagem passou a ser usada também para síntese e simulação de circuitos integrados.
Após seu sucesso inicial, sua definição foi posta em domínio público e, então, o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) a padronizou em 1987, com atualizações em 1993, 2000, 2002 e, a última, em 2008.
Sua sintaxe possui semelhanças com as linguagens de programação Pascal e Ada. Para obter maiores informações em relação a sua sintaxe e semântica, veja [3].
Atualmente é uma das linguagens mais utilizadas em sistemas de desenvolvimento de software/hardware para FPGA, como ISE da Xilinx [26], Quartus II da Altera [12], Diamond da Lattice [13] e Libero SoC da Microsemi [14].

2.4 FPGA

FPGA (*Field Programmable Gate Array*) é um circuito integrado que permite o desenvolvimento de projetos digitais após sua manufatura e que tolera reprogramações das conexões de suas estruturas e conteúdos de tabelas e, que geralmente, são realizadas com linguagens de descrição de *hardware* (*Hardware Description Language* ou HDL) [25].

Seu criador foi Ross Freeman, que era um funcionário da Zilog e queria dispostivos que funcionassem como uma fita magnética em branco para que o usuário pudesse programar a sua própria arquitetura. Porém, seria necessária uma grande quantidade de transistores, um recurso que, no começo da década de 80, era extremamente caro e limitado. Após a tentativa frustrada de convencer a Zilog em investir nesta tecnologia, Freeman e James V. Barnett II sairam desta empresa e uniram-se com Bernard V. Vonderschmitt, um outro ex-funcionário, para arrecadar 4,5 milhões de dólares e desenvolver o primeiro FPGA. Então, em 1984, os três criaram a Xilinx e, em 1985, começaram a vender seus primeiros produtos.
Em 2010, a Xilinx detinha 49% do mercado e a Altera, 40%. Logo, ambas formavam um duopólio de 89%. O restante estava nas mãos da Lattice Semiconductors, Microsemi (atual proprietária da Actel) e Quicklogic, que possuiam, respectivamente 6%, 4% e 1% da fatia desse mercado [32].

2.5 Implementações do algoritmo de Canny em plataformas FPGA

Na literatura acadêmica, a quantidade de implementações envolvendo o algoritmo de Canny com a tecnologia FPGA é grande como é visto a seguir.
Trost, Zemva e Zajc [53] descrevem uma implementação em FPGA muito similar ao algoritmo clássico descrito por Canny, porém, omitem o algoritmo utilizado para a histerese. Peng *et al.* [44] propõem algumas mudanças como o uso do filtro de mediana no lugar do filtro gaussiano, aplicam o algoritmo SHOPV (*Second Harmonic of the Variable Parameters*) para o cálculo do gradiente e calculam a histerese apenas com a vizinhança do pixel, sem percorrer as bordas. Muthukumar e Rao [40] propõem uma implementação semelhante ao [44], porém, o cálculo da histerese percorre as bordas e é realizado em um pixel por vez, sem paralelismo. Desmouliers *et al.* [15] também propõem a mesma arquitetura e utilizam o arcabouço PICO para o desenvolvimento, que faz a conversão de programas na linguagem C para sistemas em nível de *hardware* [5]. Houari e Cherrad [24] atém-se ao algoritmo clássico, utilizam uma máscara gaussiana 3x3 e o cálculo da intensidade do gradiente é realizado através da soma dos valores absolutos da derivada primeira. A histerese é igual ao [40].
He e Yuan [22] descrevem uma implementação no FPGA Altera Cyclone (chip EP1C60240C8), os limiares são autoadaptativos, a histerese não percorre as bordas (apenas a vizinhança de cada pixel) e o cálculo do gradiente é feito apenas com o valor máximo das adjacências. Xu, Chakrabarti e Karam [54, 55] propõem o algoritmo de forma distribuída com sobreposição de blocos, limiares autoadaptativos baseados no histograma da imagem e histerese idêntica ao [22]. Li, Jiang e Fan [34] também

publicaram algo semelhante. Mas, na fase de suavização é aplicado o filtro de mediana antes de aplicar a convolução gaussiana. A histerese também é idêntica ao [22].
Luk, Wu e Page [37] propõem uma implementação no FPGA CHS2x4 e utilizam a estratégia de dividir os dados, calcular e, então juntar os resultados. A implementação citada utiliza quatro blocos que são invocados um por vez através da reconfiguração dinâmica da lógica de controle. Lorca, Kessal e Demigny [35] propõem uma implementação em FPGA e em um circuito integrado dedicado (ASIC) do algoritmo de Canny-Deriche. Pankeiwicz, Owiertowski e Roszak [43] propõem uma implementação em VHDL de um detector de bordas, cujo algoritmo é uma mescla do algoritmo de Canny com a transformada de Hough. Gentsos *et al.* [17] propõem uma implementação em pipeline do Canny e utilizam BRAMs (*Block Rams*), que são blocos distribuídos de memória RAM construídos com *Lookup Tables* [11], como cache entre os módulos do pipeline. O paralelismo é explorado no fase de suavização e do operador Sobel, realizando a convolução com 4 pixeis por vez. A histerese utiliza 3 pixels de vizinhança para avaliação de bordas, ao invés dos 8 vizinhos. Neoh e Hazanchuk [42] propõem o uso de pilha de dados como uma forma de otimização da histerese. Gibson *et al.* [18] empregam técnicas estatísticas para a detecção de bordas. Amaricai *et al.* [2] propõem uma solução mista. Da suavização até a supressão de não-máximos é realizado em hardware FPGA e a histerese, em software. Haghi, Sheikh e Marsono [21] também desenvolveram algo similar. Kelefouras, Kritikakou e Goutis [33] implementam o Canny em uma Virtex-5 utilizando o processador Microblaze com uma arquitetura otimizada para a memória do FPGA. Chaithra e Ramana Reddy [10] implementaram o algoritmo de Canny no FPGA Xilinx Spartan-3E com uma arquitetura em *pipeline*, com blocos de memória e não explora o paralelismo entre os pixeis.
Nenhuma das publicações citadas empregam a arquitetura em *pipeline* com paralelismo entre os pixeis na plataforma FPGA, todas apresentam o algoritmo implementado em VHDL, de forma análoga ao caso de emprego de outras linguagens de programação. Com isto, não exploram o paralelismo que pode ser obtido com arquiteturas desenvolvidas com recursos de *pipeline* e processamento matricial, por exemplo.

2.6 Barramento PCI-Express

A plataforma FPGA foi instalada em um computador hospedeiro e a troca de dados ocorre por meio de uma conexão a um barramento de entrada e saída. A plataforma FPGA utilizada possui interface para o PCI-Express (*Peripheral Component Interconnect Express* - sigla oficial PCIe) que é um barramento serial de alta velocidade que foi criado para substituir os padrões PCI, PCI-X e AGP [20]. A principal diferença entre o PCI e o PCIe está em suas topologias. O primeiro possui um barramento compartilhado onde todos os *slots* de uma placa-mãe compartilham o mesmo barramento

para comunicar-se com o *bridge*. No caso do PCIe, a conexão é ponto-a-ponto com o *bridge* da placa-mãe. Um *slot* PCIe pode ter 1, 2, 4, 8, 16 e 32 caminhos de dados, que, na documentação oficial, são identificados pelo sufixo *x*. Por exemplo, um *slot* com 8 caminhos é identificado por 8x. Cada caminho de dados, possui uma taxa de transferência de 250 MB/s na versão 1.1 do padrão e 500 MB/s na versão 2.0.
O protocolo de comunicação deste barramento é similar a um protocolo de rede. A comunicação entre um dispositivo com PCIe e um processador só é realizada através de troca de pacotes e o protocolo é composto pelas seguintes camadas:

- Física: camada que define as especificações elétricas e da lógica digital;
- Enlace de Dados: camada que define o sequenciamento dos pacotes, verificação de erros e retransmissão de dados em caso de erros;
- Transmissão: camada que faz a interface do protocolo com o usuário final.

Neste trabalho é utilizado este barramento devido a sua taxa de transferência que, no caso da plataforma disponível, é de 5.0 Gb/s para 4x e 2.5 Gb/s para 8x.

2.7 PIO

A escolha do barramento PCI-Express foi por causa da alta taxa de transferência e pela simplicidade do protocolo PCI-Express em relação ao protocolo Ethernet. A Xilinx, através de sua plataforma de desenvolvimento, oferece duas maneiras de implementação em FPGA utilizando o barramento PCI-Express (PCIe). Estas são as seguintes:

- Programmed Input/Output (PIO): é um método de transferência de dados entre dispositivos que utiliza o processador do computador como parte do caminho de dados (*datapath*). Sua principal vantagem é a comunicação direta entre o PCIe e a lógica desenvolvida no FPGA e a sua desvantagem é a falta de portabilidade da lógica desenvolvida para outras arquiteturas;
- Advanced eXtensible Interface (AXI): é a terceira geração da interface AMBA (*Advanced Microcontroller Bus Architecture*) que é um barramento desenvolvido para processadores ARM visando arquiteturas SoC (*System on Chip*). Sua aplicação visa sistemas de tempo-real para o processamento de arquivos de video (*streaming*). Sua principal vantagem encontra-se na portabilidade da lógica desenvolvida que pode ser transposta para outros dispositivos que suportam tal interface como, por exemplo, arquiteturas PSoC (*Programmable System on Chip*). Todavia, sua desvantagem está no uso maior de espaço em *hardware* do que o PIO para suportá-lo no FPGA, pois, esta interface acaba usando também o PIO para realizar a comunicação com o PCIe.

Por ocupar menos espaço, foi escolhido o PIO como forma de interface com o FPGA. A arquitetura do PIO para PCIe disponibilizado pela Xilinx é mostrada na Figura 2.1. A implementação simula uma memória dentro do FPGA.

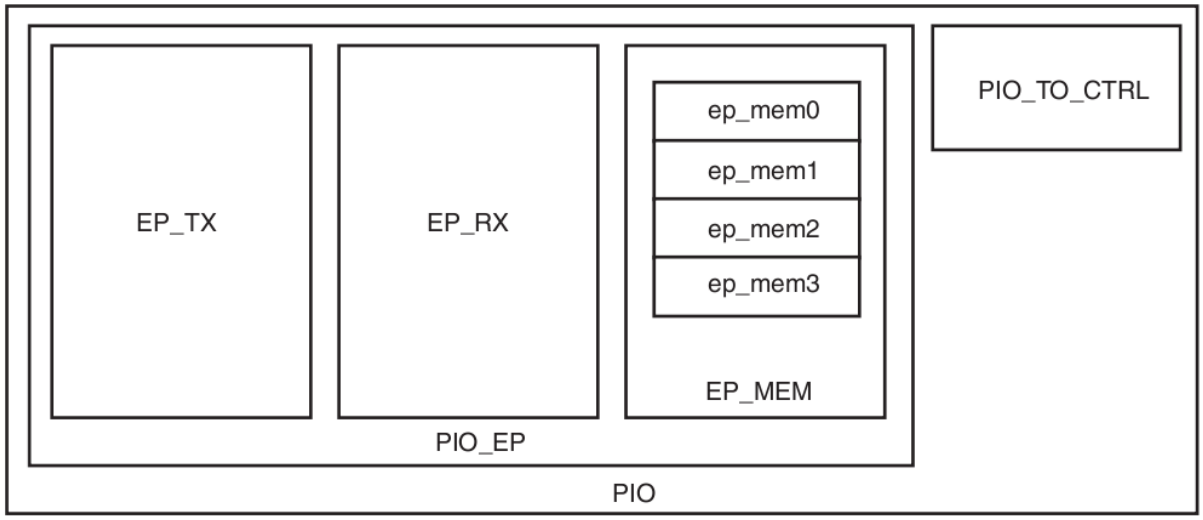

Figura 2.1: Entidades da arquitetura PIO para PCIe disponibilizada pela Xilinx (Fonte: [27])

A arquitetura é composta pelas seguintes entidades [27]:

- PIO: esta encapsula e instancia todas as subentidades internas, exceto o controle;
- PIO_TO_CTRL: é o responsável pela sincronização da comunicação. Quando o processador envia a mensagem de término, esta entidade retorna uma mensagem de aceitação (ACK);
- PIO_EP: esta encapsula as entidades de memória e comunicação com o barramento PCIe;
- EP_RX: é responsável pela gerência e manuseio dos pacotes recebidos pelo barramento PCIe e entrega os dados para o entidade EP_MEM;
- EP_TX: é responsável pelo recebimento dos pacotes do entidade EP_MEM e as transmite para barramento PCIe;
- EP_MEM: é gerenciadora das memórias ep_mem0, ep_mem1, ep_mem2 e ep_mem3;
- ep_mem0, ep_mem1, ep_mem2 e ep_mem3: são os bancos de memória criados dentro do FPGA.

2.8 Plataforma de desenvolvimento Xilinx Virtex-6

A plataforma em uso é a *Virtex-6 FPGA ML605 Evaluation Kit* da Xilinx [28] (Figura 2.2). Esta possui uma gama bem diversificada de hardware e ferramentas de desenvolvimento. O chip FPGA desta plataforma é o XC6VLX240T-1FFG1156 que possui 241.152 células lógicas, 37.680 *slices*, cada *slice* é composto por 4 *look-up tables* (LUT) e 8 *flip-flops*, e 14.976 Kbits de RAM de blocos (*block RAM*), cada bloco pode ter 18 ou 36 Kbits de espaço para armazenamento [29]. A plataforma possui as seguintes interfaces:

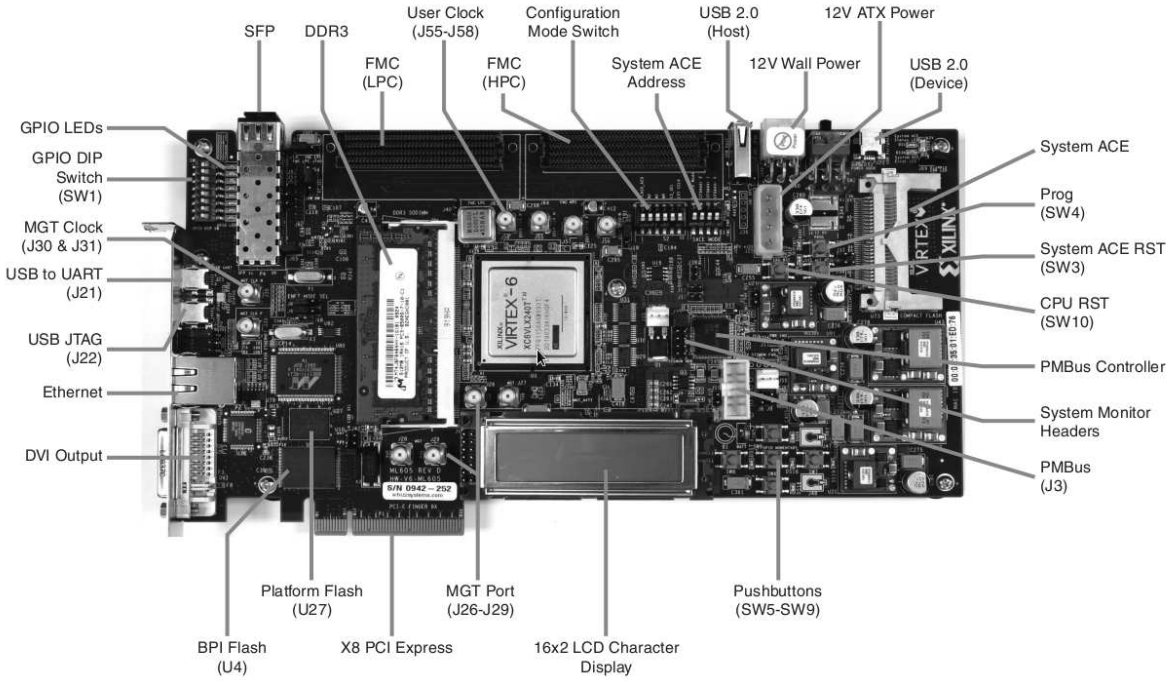

Figura 2.2: Virtex-6 FPGA ML605 Evaluation Kit (Fonte: xilinx.com)

- saída de vídeo DVI (*Digital Visual Interface*);
- memória DDR3 (*Double Data Rate type Three*) de 512 Megabytes;
- porta de Entrada e Saída GPIO (*General-purpose input/output*);
- porta de conexão Ethernet 10/100/1000 Mb/s;
- porta para Fibra ótica;
- suporte a memória do tipo *Compact Flash*;
- suporte ao barramento PCI-Express Gen1 (x8) e Gen2 (x4);
- suporte aos barramentos USB 2.0, USB JTAG e USB para UART.

Para realizar a síntese nesta plataforma é necessário o ambiente de desenvolvimento Xilinx ISE (*Integrated Software Environment*) [26]. Este ambiente é composto dos seguintes aplicativos:

- Xilinx Platform Studio (XPS): ferramenta que permite construir e integrar os módulos desesenvolvidos;
- Xilinx CORE Generator System (coregen): ferramenta que provê um conjunto de módulos personalizáveis para o uso dos recursos do FPGA e seus periféricos;
- Bitgen: ferramenta que permite a criação de arquivos binários compatíveis com o FPGA (com o sufixo .bit) através do código-fonte;
- Xilinx iMPACT: ferramenta que permite transpor o arquivo binário para o FPGA.

Os sistemas operacionais oficialmente suportados e homologados pelo fabricante são o Red Hat Enterprise versões 4,5 e 6, Suse Linux Entrerprise 11 e Microsoft Windows versões XP e 7 Professional. Neste trabalho é utilizado a distribuição CentOS (http://www.centos.org) que é uma versão comunitária baseada no Red Hat Enterprise. Sua adoção foi realizada devido a sua grande similaridade com o Red Hat Enterprise (distribuição homologada) e por ser gratuita.

CAPÍTULO 3

PROJETO

Este capítulo tem como objetivo descrever toda a implementação do projeto, tanto a parte que encontra-se no computador PC como a parte que está no FPGA. Desta última parte serão descritos também interface e o funcionamento de cada entidade alí existente, com as suas conexões e relação entre essas. Este capítulo possui as seguintes seções:

- Seção 3.1 - Visão do Projeto: descreve a visão global deste projeto;
- Seção 3.2 - Implementação contida no PC: descreve a parte do algoritmo de Canny que foi implementada no PC;
- Seção 3.3 - Implementação contida no FPGA: descreve a parte do algoritmo de Canny que foi implementada no FPGA.

3.1 Visão do Projeto

A idéia inicial deste projeto era implementar todo o algoritmo de Canny no FPGA, mantendo a arquitetura massivamente paralela e do tipo *pipeline*, o que não foi possível devido a área ocupada pelo binário no FPGA, como corrrobora o anexo A.1. Logo, a solução encontrada foi a plataforma híbrida, parte no computador PC e parte no FPGA, que é diferente da solução proposta por Amaricai *et al.* [2], cujas etapas de suavização, cálculo do gradiente e supressão de não máximos encontram-se no FPGA e a histerese encontra-se no PC. A solução proposta aqui é dividida de forma praticamente oposta, com a exceção da primeira fase da histerese, que também foi mantida no PC. A Figura 3.1 ilustra a divisão do trabalho proposto, do que vai no PC e do que vai no FPGA.

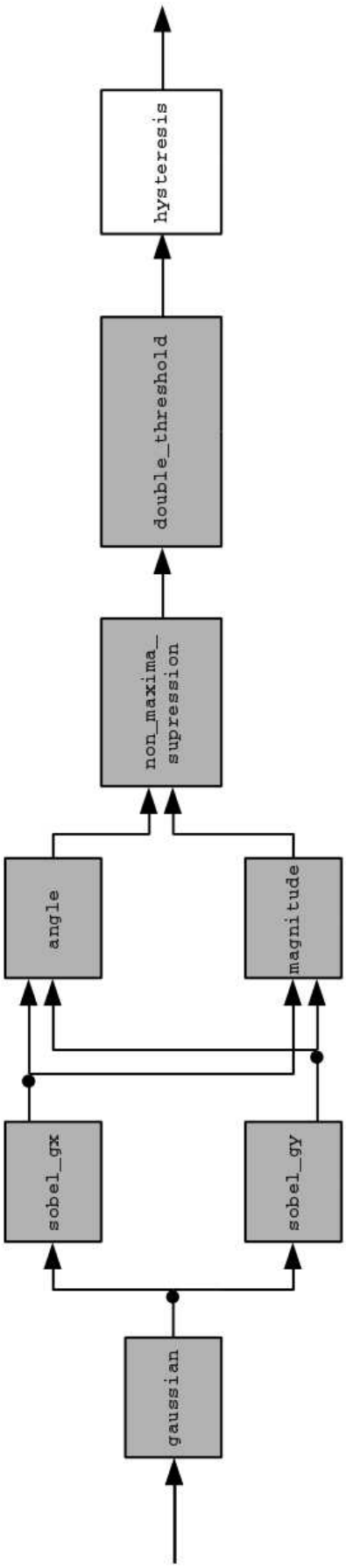

Figura 3.1: Diagrama de blocos que ilustra as partes implementadas no PC (blocos com a cor cinza) e no FPGA (bloco com a cor branca) (Figura do autor).

Esta divisão foi interessante para o caso da arquitetura proposta porque cada etapa do algoritmo de Canny, com exceção da segunda fase da histerese, ocupa uma quantidade grande da área do FPGA, como corrobora o anexo A.2. Outro fator interessante é que a segunda fase do algoritmo de histerese é a etapa mais custosa do algoritmo de Canny (Subseção 3.3.1). Portanto, o desenvolvimento no FPGA torna-se atrativo para o ganho de desempenho.

3.2 Implementação contida no PC

A parte contida no PC foi baseada na função *Canny* da biblioteca OpenCV [7], implementada com a linguagem C++ [52]. Foram realizadas alterações com o intuito de remover a segunda fase da histerese e retornar resultados que possam ser interpretados pelo módulo que se encontra no FPGA. O código encontra-se no Anexo B.1 (função *almost_Canny*).
Outra função do OpenCV utilizada foi *GaussianBlur* pois esta é uma etapa importante e consta no algoritmo original [9], cujo o objetivo é eliminar ruídos da imagem (suavização) e evitar bordas falsas. O *GaussianBlur* não se encontra dentro da função *Canny* do OpenCV. Neste módulo também não foi embutida a função de suavização dentro da função *almost_Canny* (Anexo B.1). Mas, *GaussianBlur* sempre é executada antes de *Canny* e *almost_Canny* no PC. O valor de desvio padrão da máscara utilizada na implementação é de 1,4 (Figura 3.2). Caso a máscara seja aplicada em cantos, onde não há pixeis, é utilizado o valor do próprio pixel do canto.

$$\text{Saida} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \text{Entrada}$$

Figura 3.2: Convolução gaussiana da implementação proposta (Figura do autor).

Na etapa do cálculo da primeira derivada nas direções x e y é realizado pela função *Sobel* do OpenCV. As máscaras aplicadas nas direções x e y estão expressas nas Figuras 3.3a e 3.3b, respectivamente [51].

$$G_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_Y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

(a) Direção x (G_x) (b) Direção y (G_y)

Figura 3.3: Máscaras do operador Sobel nas direções x e y (Figuras do autor).

Quanto às etapas de cálculo do ângulo e magnitude, supressão de não máximos e a primeira fase da histerese, cujo o algoritmo é explicado de forma mais detalhada na Seção 3.3.1, são todos calculados no mesmo laço de execução, ao invés de ser um laço para cada etapa. Quanto ao cálculo de magnitude, a implementação oferece duas formas de equação. A primeira é uma aproximação da fórmula original que é a soma dos valores absolutos de G_x e G_y (Equação 3.1) e a segunda é exatamente a fórmula original (Equação 3.2).

$$\boxed{M = |G_x| + |G_y|} \tag{3.1}$$

$$\boxed{M = \sqrt{G_x^2 + G_y^2}} \tag{3.2}$$

3.3 Implementação contida no FPGA

Nesta seção é descrita o algoritmo de histerese adotado e a entidade *hysteresis* contida no FPGA, como também são descritas as entidades internas a *hysteresis* nas seções à seguir.

3.3.1 Algoritmo de Histerese

Esta etapa é a mais custosa e a mais complexa do algoritmo de Canny, pois o término do algoritmo da histerese depende dos limiares escolhidos e dos pontos pertencentes à alguma borda. Há diversas abordagens para este algoritmo [1, 9, 36, 39, 47, 50, 56]. A abordagem usada na implementação proposta é similar ao [36], que consiste em separar inicialmente os pixels nas classes eleito, não eleito e candidato (Algoritmo 1). Depois, há uma segunda etapa onde cada pixel é processado individualmente. Se o pixel é candidato e possui algum vizinho eleito, então este torna-se eleito. O término da execução do algoritmo ocorre quando não existe um pixel candidato com um vizinho eleito (Algoritmo 2).

Algoritmo 1 Algoritmo da primeira fase da histerese

```
for all Pixel ∈ Imagem do
  if Pixel > Limite Superior then
    Label ← Eleito
  else if Pixel < Limite Inferior then
    Label ←Não Eleito
  else
    Label ← Candidato
  end if
end for
```

Algoritmo 2 Algoritmo da segunda fase da histerese

```
repeat
  for all Label ∈ Imagem do
    if Label = Candidato then
      if Existe algum vizinho de Pixel que seja Eleito then
        Label ← Eleito
      end if
    end if
  end for
until Não exista nenhum Pixel Candidato com algum vizinho Eleito
```

A primeira fase deste algoritmo encontra-se no PC e foi deixada lá por ocupar muito espaço no FPGA e por não ser uma etapa custosa. Já a segunda fase foi implementada em FPGA para fins de desempenho e por ser, de fato, a etapa mais custosa do processo de detecção de bordas.

3.3.2 A entidade *hysteresis*

A entidade *hysteresis* (Figura 3.4) faz a interface com a arquitetura PIO (*Programmed Input/Output*) que é um método de transferência de dados entre dispositivos e utiliza o processador do computador como parte do caminho de dados (*datapath*). O código-fonte encontra-se no Anexo E.3. Todas as entidades usam o código-fonte *work_type.vhdl* como cabeçalho.
Esta entidade recebe os seguintes dados como entrada:

- Oito *bits* que compõem um pixel da imagem (*pixel_in*);
- Oito *bits* para identificação da imagem (*tag_in*);
- Dezesseis *bits* que compõem o endereço de *pixel_in* (*addressi_in_w*);
- Dezesseis *bits* que compõem o endereço para obtenção do dado de *pixel_out* (*address_in_r*);

- Um sinal para colocar a entidade em seu estado inicial (*reset*);
- Um sinal para sincronizar a evolução de estados da entidade (*clk*);

A entidade possui as seguintes saídas:

- Oito *bits* que compõem um pixel da imagem de saída (*pixel_out*);
- Oito *bits* para identificação da imagem de saída (*tag_out*);
- Um sinal para indicar a saída do primeiro pixel da imagem processada (*start*);
- Um sinal para indicar a saída do último pixel da imagem processada (*finish*).

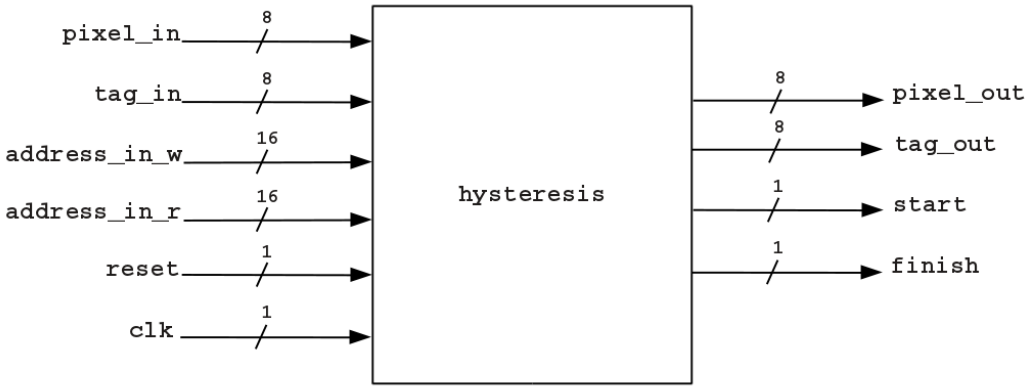

Figura 3.4: Entidade *hysteresis* (Figura do autor)

Quanto à sua estrutura interna, esta tem como base as entidades *serial_to_parallel*, *hysteresis_pipeline*, *parallel_to_serial* e *bubble_tag*. As entidades *serial_to_parallel* e *parallel_to_serial* servem para realizar a interface de *hysteresis_pipeline* com a entrada e a saída da interface do PCI-Express, que é serial. A paralelização da entrada de *hysteresis_pipeline* é útil para explorar o paralelismo entre os rótulos (*label*) da imagem. A Figura 3.5 mostra como essas entidades estão interligadas. Foram omitidos alguns sinais neste diagrama por questões de clareza.

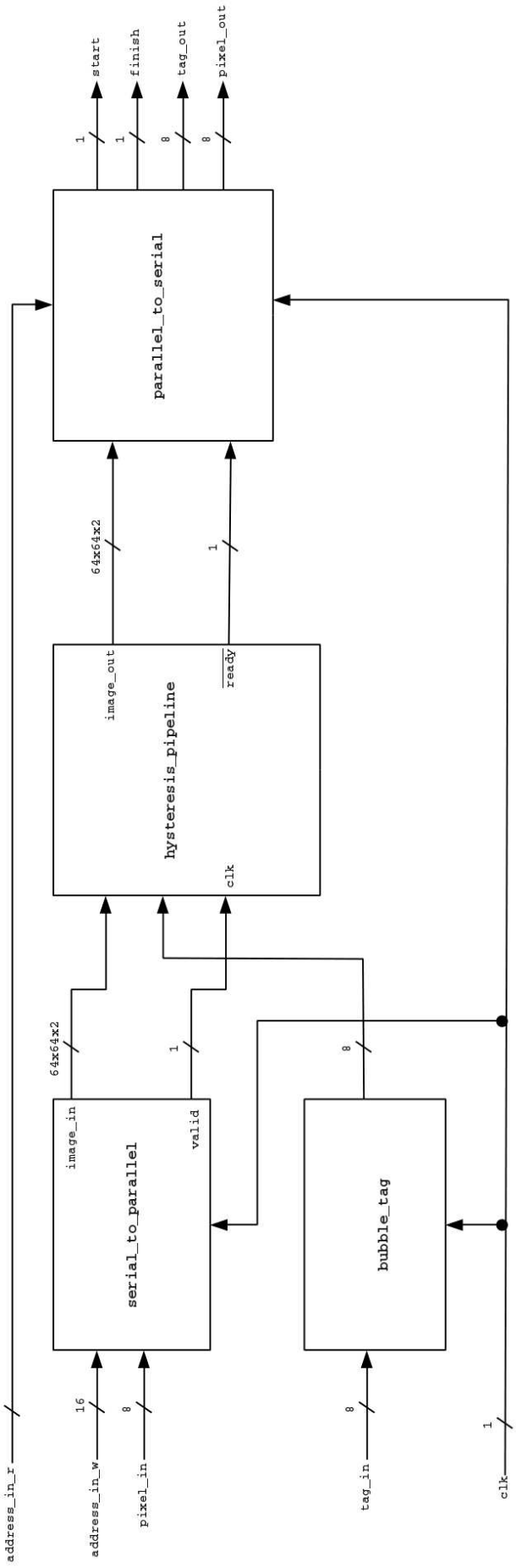

Figura 3.5: Estrutura interna da entidade *hysteresis* (Figura do autor)

3.3.3 A entidade *serial_to_parallel*

Esta entidade tem como função converter os valores de 8 bits que entram em *pixel_in*, de acordo com o endereço recebido em *address_in*, para um sinal de 64 x 64 elementos com 2 bits cada (*image_out*), cujo sinal *ready* indica que há uma matriz de rótulos válida na saída de *image_out*. Esta tem em comum com as outras entidades os sinais de entrada *reset* e *clk* (Figura 3.6). O código-fonte encontra-se no Anexo E.19.

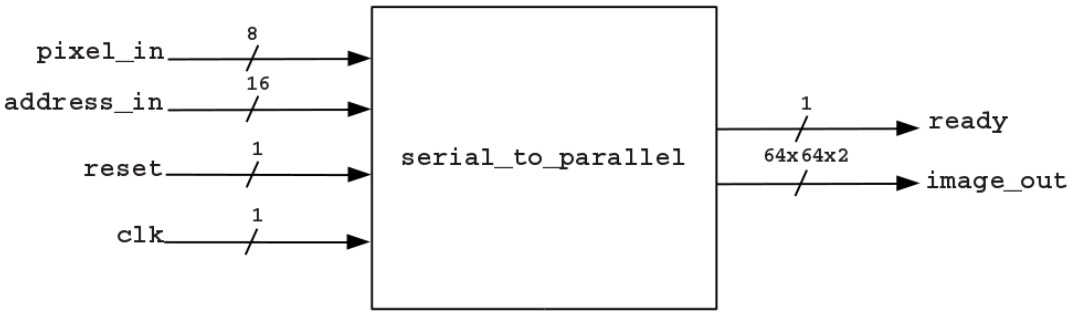

Figura 3.6: Interface de *serial_to_parallel* (Figura do autor)

3.3.4 A entidade *parallel_to_serial*

Esta entidade tem a função inversa de *serial_to_parallel*: recebe um sinal de 64 x 64 elementos com 8 bits cada e envia esses pixeis, de acordo com o endereço recebido em *address_in*, para a saída *pixel_out*. O sinal *retain* serve para reter os dados que estão internamente presentes, não permitindo a gravação dos dados de entrada, caso seja '1'. Esta entidade possui um conjunto de entradas e saídas muito semelhante à entidade *hysteresis*. Possui uma similaridade nas entradas *image_in*, *tag_in*, *reset* e *clk*. E, nas saídas *tag_out* *start* e *finish* (Figura 3.7). O código-fonte encontra-se no Anexo E.18.

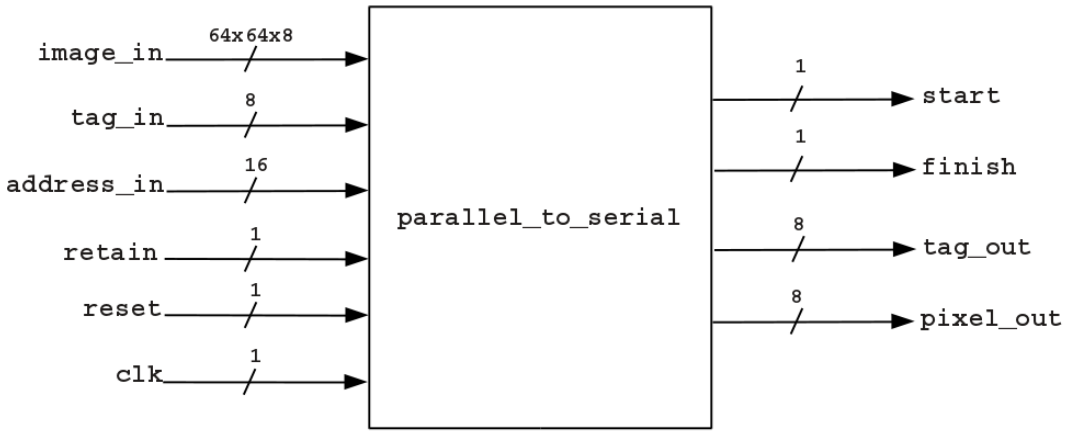

Figura 3.7: Interface de *parallel_to_serial* (Figura do autor)

3.3.5 A entidade *bubble_tag*

Esta entidade serve para propagar o identificador da imagem, que é um valor de 8 *bits*. Tem em comum com as outras entidades os sinais de entrada *retain*, *reset* e *clk*. Além destas, possui como entrada um valor de 8 *bits* (*tag_in*) e outro como saída (*tag_out*) (Figura 3.8).

Figura 3.8: Interface de *bubble_tag* (Figura do autor)

3.3.6 A entidade *hysteresis_pipeline*

A entidade *hysteresis_pipeline* é a implementação da segunda etapa do algoritmo da histerese neste projeto. Esta recebe os seguintes dados como entrada:

- *image_in*: é um sinal de 64 x 64 elementos com 2 bits cada, que é a matriz de rótulos da parte processada pelo PC;
- *tag_in*: é o identificador da imagem de entrada;
- *reset*: é o sinal que coloca a entidade em seu estado inicial;
- *clk*: é o sinal de sincronização da entidade;

E, possui as seguintes saídas:

- *image_out*: é um sinal de 64 x 64 elementos com 8 *bits*, que são os pixeis da imagem processada pela entidade;
- *tag_out*: é o identificador da imagem de saída;
- $\overline{ready}$: é o sinal que indica quando a imagem de saída está pronta. O sinal '0' (zero) indica que *image_out* tem uma imagem pronta.

A Figura 3.9 é uma representação dessa entidade com suas entradas e saídas.

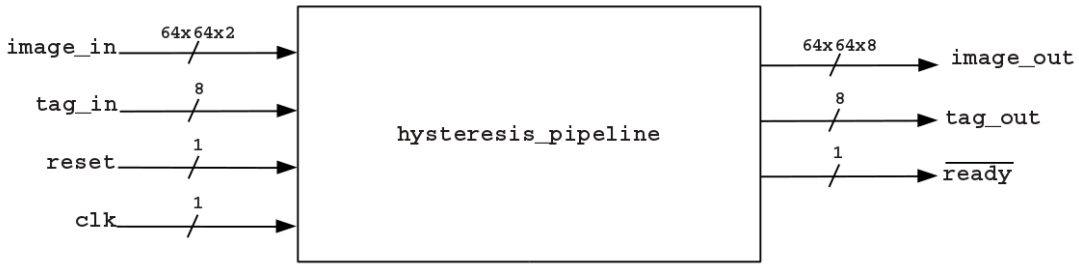

Figura 3.9: Interface de *hysteresis_pipeline* (Figura do autor)

A estrutura interna da *hysteresis* tem como base as entidades *bubble*, *bubble_tag*, *hysteresis_check_pixel*, *hysteresis_out*, *hysteresis_spread_elect*, *hysteresis_sum_first*, *hysteresis_sum_second*, *hysteresis_sum_third*, *mux_tag* e *mux*. A Figura 3.10 mostra como essas entidades estão interligadas. Foram omitidos os sinais *reset* e *clk* neste diagrama por questões de clareza, mas, subendenda-se que esses estão ligados com todas as entidades. O código-fonte encontra-se no Anexo E.8.

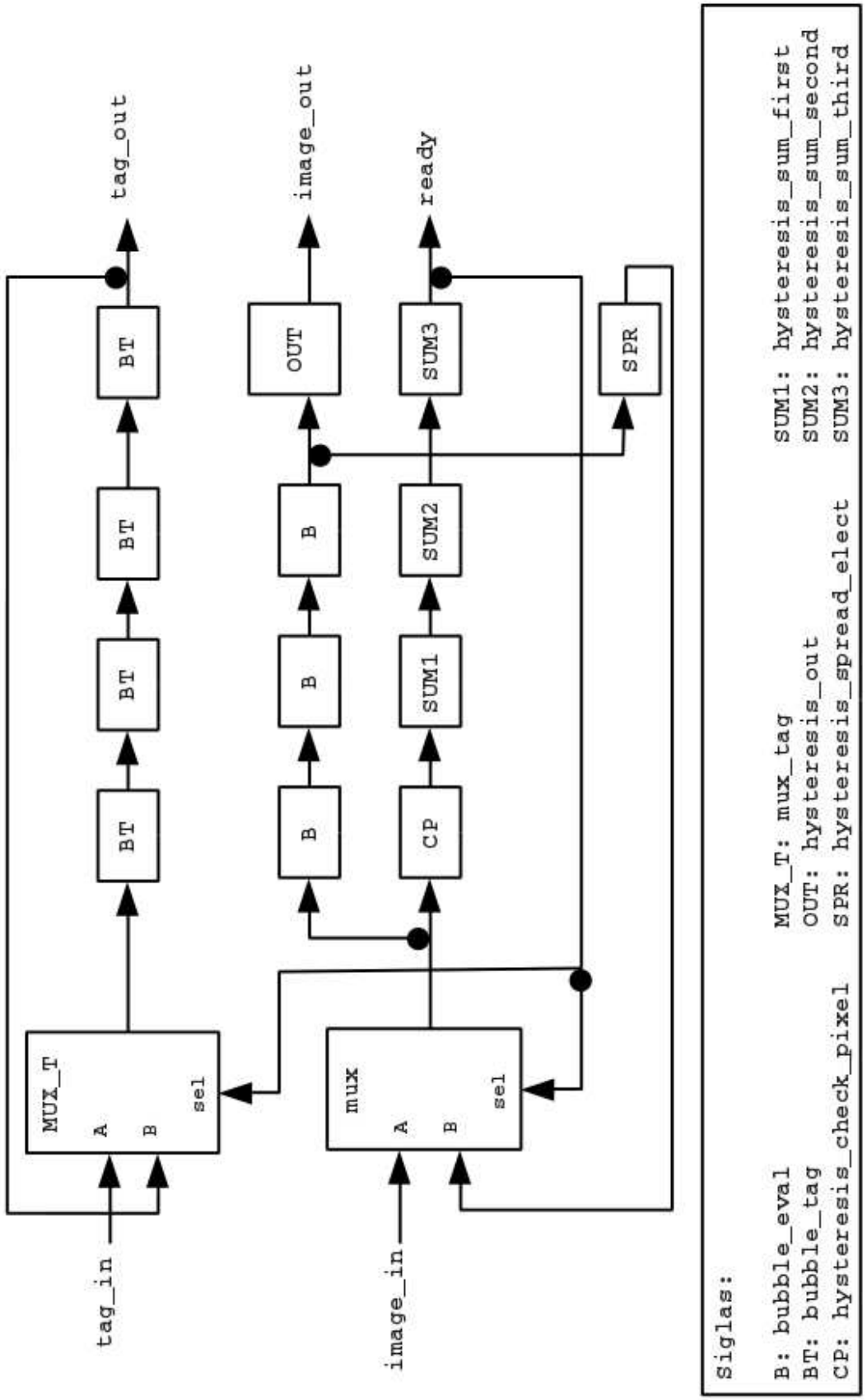

Figura 3.10: Estrutura interna da entidade *hysteresis_pipeline* (Figura do autor)

As subseções a seguir tem como objetivo descrever as entidades internas a *hysteresis_pipeline*.

3.3.7 As entidades *bubble_eval* e *bubble_eval_element*

A entidade *bubble_eval* serve para propagar a matriz de rótulos a ser processada, que é um sinal de 64 x 64 elementos com 2 *bits*. *image_in* é a matriz de rótulos de entrada e *image_out* é a matriz de rótulos de saída. *bubble_eval* tem em comum com as outras entidades os sinais de entrada *reset* e *clk* (Figura 3.11). O código-fonte encontra-se no Anexo E.1.

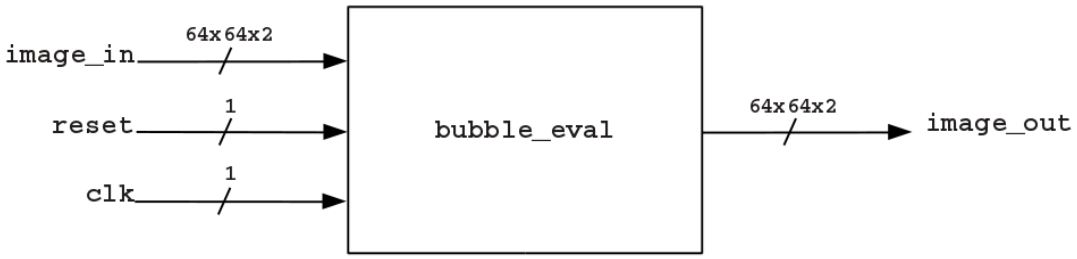

Figura 3.11: Interface de *bubble_eval* (Figura do autor)

Internamente, a entidade *bubble_eval* possui 64 x 64 instâncias de *bubble_eval_element*, no qual cada elemento armazena 2 *bits* que correspondem a um rótulo de pixel. *image_in* é a matriz de rótulos de entrada e *image_out* é a matriz de rótulos de saída. *bubble_eval* tem em comum com as outras entidades os sinais de entrada *reset* e *clk* (Figura 3.12). O código-fonte encontra-se no Anexo E.2.

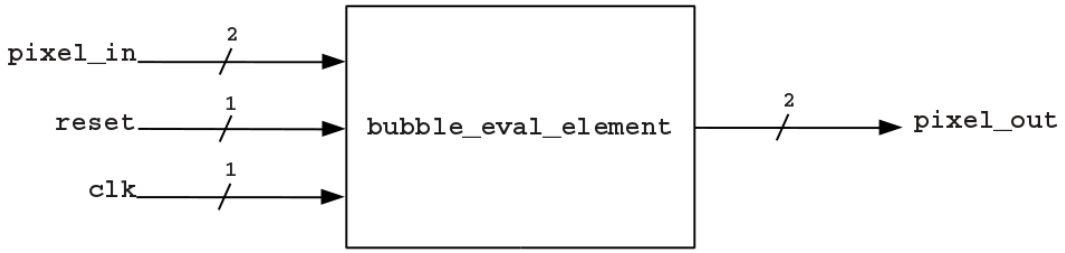

Figura 3.12: Interface de *bubble_eval_element* (Figura do autor)

3.3.8 As entidades *mux*, *mux_element* e *mux_tag*

A entidade *mux* serve como um multiplexador para duas matrizes de rótulos (*image_in_a* e *image_in_b*). Caso o sinal *sel* seja igual a '0' (zero), a imagem em *image_in_a* sairá em *image_out*. Sairá *image_in_b* em *image_out*, caso *sel* seja '1' (um). Internamente, *mux* possui uma matriz com 64 x 64 instâncias de *mux_element*. Possui em comum com a

entidade *bubble_eval* apenas as entradas *reset* e *clk*, e a saída *image_out*. O código-fonte encontra-se no Anexo E.15 Além dessas, possui também as seguintes entradas (Figura 3.13):

- *image_in_a*: é um sinal de 64 x 64 elementos com 2 bits cada, que é a matriz de rótulos que irá para *image_out* caso *sel* seja igual a '0' (zero);
- *image_in_b*: é um sinal de 64 x 64 elementos com 2 bits cada, que é a matriz de rótulos que irá para *image_out* caso *sel* seja igual a '1' (um);
- *sel*: é o sinal de seleção, cujo valor indica qual matriz de rótulos sairá em*image_out*.

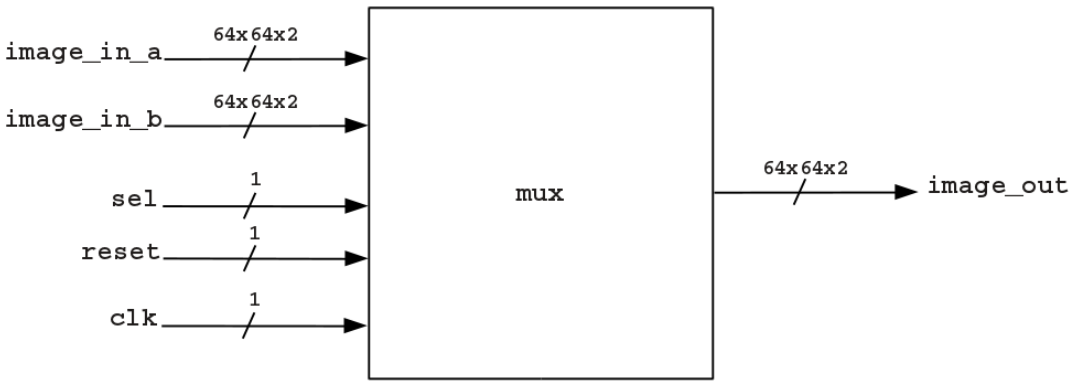

Figura 3.13: Interface de *mux* (Figura do autor)

A interface de *mux_element* está expressa na figura 3.14. As entradas *sel*, *reset* e *clk* de *mux_element* são idênticas a *mux*. *pixel_in_a* e *pixel_in_b* são os rótulos de pixel que serão selecionados de acordo com *sel* e sairão em *pixel_out*.

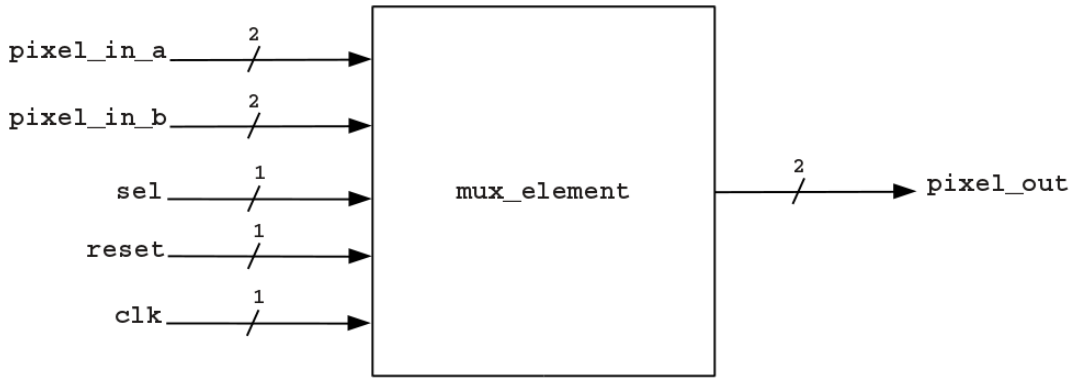

Figura 3.14: Interface de *mux_element* (Figura do autor)

O algoritmo 3, em pseudocódigo, descreve a funcionalidade de *mux_element*. O código-fonte em VHDL encontra-se no Anexo E.16.

Algoritmo 3 Algoritmo da entidade *mux_element*

```
if sel =' 0' then
  pixel_out ← pixel_in_a
else
  pixel_out ← pixel_in_b
end if
```

A interface de *mux_tag* está expressa na figura 3.15. As entradas *sel*, *reset* e *clk* de *mux_tag* são idênticas a *mux*. *tag_in_a* e *tag_in_b* são os identificadores de entrada que serão selecionados de acordo com *sel* e sairão em *tag_out*.

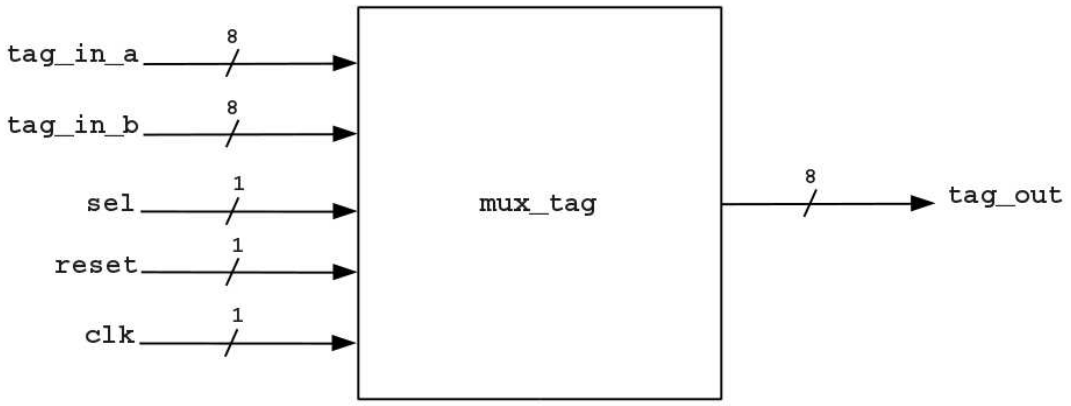

Figura 3.15: Interface de *mux_tag* (Figura do autor)

O algoritmo 4, em pseudocódigo, descreve a funcionalidade de *mux_tag*. O código-fonte em VHDL encontra-se no Anexo E.17.

Algoritmo 4 Algoritmo da entidade *mux_tag*

```
if sel =' 0' then
  tag_out ← tag_in_a
else
  tag_out ← tag_in_b
end if
```

3.3.9 As entidades *hysteresis_check_pixel* e *hysteresis_check_pixel_element*

Nesta entidade é feita a verificação da existência de pixeis candidatos com algum vizinho eleito. Esta entidade retorna uma matriz binária com o resultado da seguinte verificação: '1' caso o rótulo de pixel seja candidato com um vizinho eleito e '0' caso contrário. Internamente, *hysteresis_check_pixel* possui uma matriz com 64 x 64 instâncias de *hysteresis_check_pixel_element*, onde cada instância é responsável pela verificação num determinado rótulo de pixel da imagem a ser processada.

Possui todas as entradas iguais a *bubble*. Porém, a saída é um barramento de 64 x 64 elementos com 1 *bit* (*bit_array_out*). A Figura 3.16 demonstra a interface desta entidade. O código-fonte encontra-se no Anexo E.4.

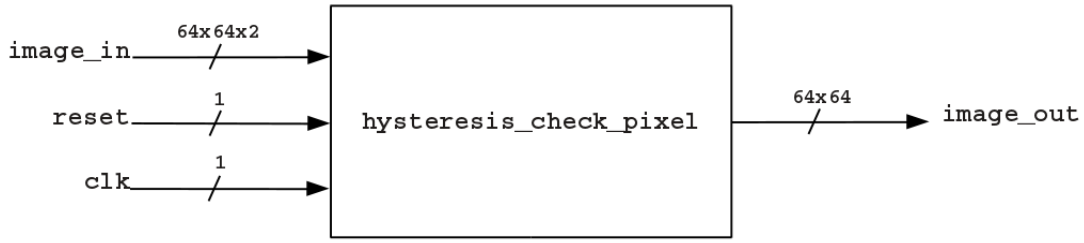

Figura 3.16: Interface de *hysteresis_check_pixel* (Figura do autor)

A interface de *hysteresis_check_pixel_element* está expressa na Figura 3.17. Tem em comum com *hysteresis_check_pixel* os sinais de entrada *reset* e *clk*. Os sinais de entrada *pixel_nw*, *pixel_ne*, *pixel_sw*, *pixel_se*, *pixel_n*, *pixel_s*, *pixel_e* e *pixel_w* são a vizinhança do pixel de entrada (*pixel_in*) nas direções noroeste, nordeste, sudoeste, sudeste, norte, sul, leste e oeste, respectivamente (Figura 3.18). *bit_out* é o pixel avaliado.

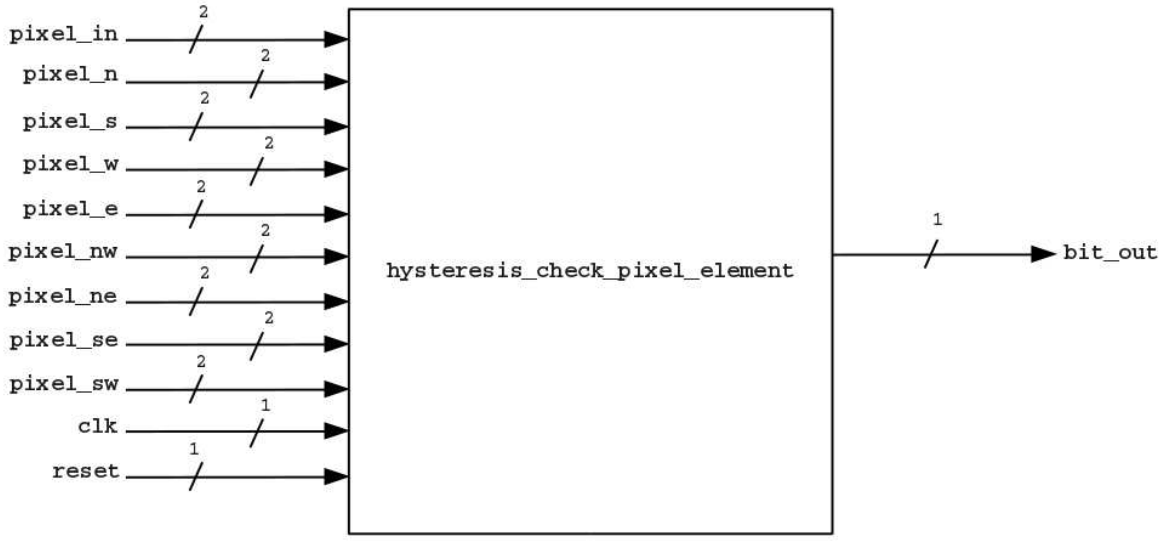

Figura 3.17: Interface de *hysteresis_check_pixel_element* (Figura do autor)

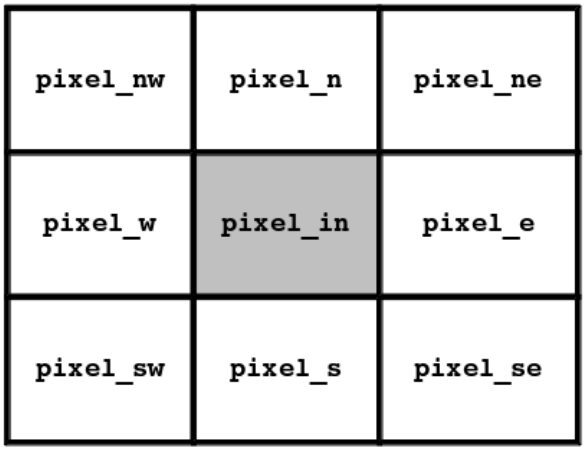

Figura 3.18: Relação entre o pixel de entrada (*pixel_in*) com seus vizinhos (Figura do autor)

O algoritmo 5, em pseudocódigo, descreve a funcionalidade de *hysteresis_check_pixel_element*. O código-fonte em VHDL encontra-se no Anexo E.5.

Algoritmo 5 Algoritmo da entidade *hysteresis_check_pixel_element*

```
if pixel_in = Candidato then
  if pixel_nw = Eleito then
    bit_out ←' 1'
  end if
  if pixel_n = Eleito then
    bit_out ←' 1'
  end if
  if pixel_ne = Eleito then
    bit_out ←' 1'
  end if
  if pixel_e = Eleito then
    bit_out ←' 1'
  end if
  if pixel_sw = Eleito then
    bit_out ←' 1'
  end if
  if pixel_s = Eleito then
    bit_out ←' 1'
  end if
  if pixel_se = Eleito then
    bit_out ←' 1'
  end if
  if pixel_w = Eleito then
    bit_out ←' 1'
  end if
else
  bit_out ←' 0'
end if
```

3.3.10 As entidades *hysteresis_spread_elect* e *hysteresis_spread_elect_element*

Nesta entidade é realizada a transformação dos rótulos de pixel marcados como candidato para eleito, caso tenham algum vizinho eleito. Internamente, *hysteresis_spread_elect* é composto por uma matriz com 64 x 64 instâncias de *hysteresis_spread_elect_element*, nas quais cada instância é responsável pela transformação de um determinado rótulo de pixel da imagem a ser processada. Possui todas as entradas e saídas iguais a *bubble_eval*. A Figura 3.19 demonstra a interface desta entidade. O código-fonte encontra-se no Anexo E.9.

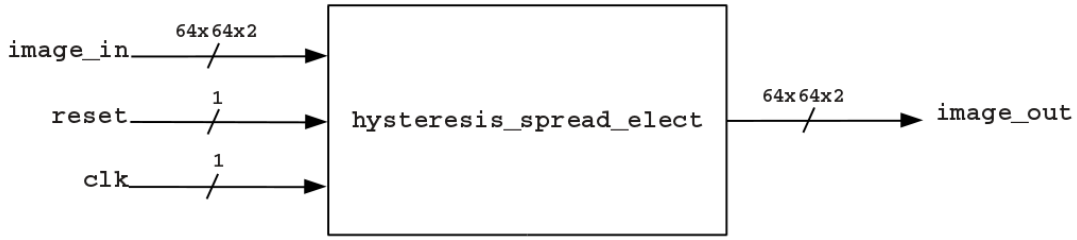

Figura 3.19: Interface de *hysteresis_spread_elect* (Figura do autor)

A interface de *hysteresis_spread_elect_element* está expressa na Figura 3.20. Tem em comum com *hysteresis_spread_elect* os sinais de entrada *reset* e *clk*. Os sinais de entrada *pixel_nw*, *pixel_ne*, *pixel_sw*, *pixel_se*, *pixel_n*, *pixel_s*, *pixel_e* e *pixel_w* são a vizinhança do pixel de entrada (*pixel_in*) nas direções noroeste, nordeste, sudoeste, sudeste, norte, sul, leste e oeste, respectivamente (Figura 3.21). *pixel_out* é o pixel transformado.

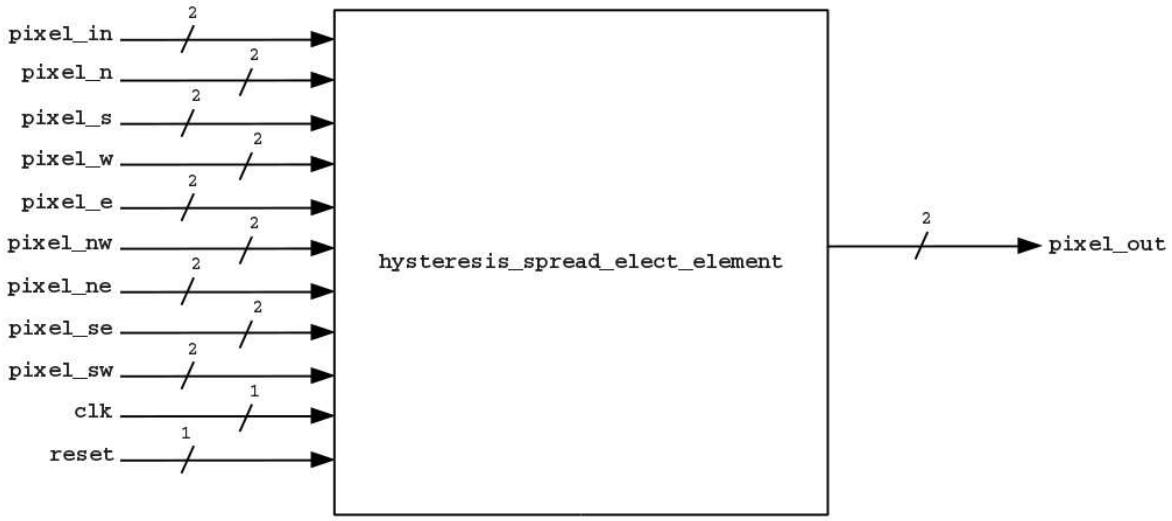

Figura 3.20: Interface de *hysteresis_spread_elect_element* (Figura do autor)

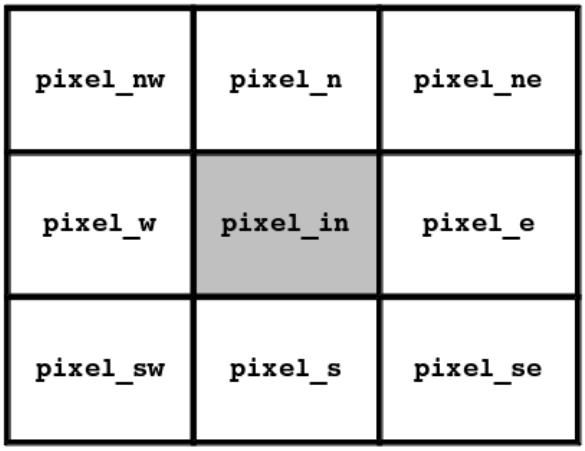

Figura 3.21: Relação entre o pixel de entrada de *hysteresis_spread_elect_element* (*pixel_in*) com seus vizinhos (Figura do autor)

O algoritmo 6, em pseudocódigo, descreve a funcionalidade de *hysteresis_spread_elect_element*. O código-fonte em VHDL encontra-se no Anexo E.10.

Algoritmo 6 Algoritmo da entidade *hysteresis_spread_elect_element*

```
if pixel_in = Candidato then
  if pixel_nw = Eleito then
    pixel_out ← Eleito
  end if
  if pixel_n = Eleito then
    pixel_out ← Eleito
  end if
  if pixel_ne = Eleito then
    pixel_out ← Eleito
  end if
  if pixel_e = Eleito then
    pixel_out ← Eleito
  end if
  if pixel_sw = Eleito then
    pixel_out ← Eleito
  end if
  if pixel_s = Eleito then
    pixel_out ← Eleito
  end if
  if pixel_se = Eleito then
    pixel_out ← Eleito
  end if
  if pixel_w = Eleito then
    pixel_out ← Eleito
  end if
else
  pixel_out ← pixel_in
end if
```

3.3.11 As entidades *hysteresis_sum_first*, *hysteresis_sum_second*, *hysteresis_sum_third* e *hysteresis_sum_element*

Dada a matriz resultante de *hysteresis_check_pixel*, é necessário realizar a soma dos elementos dessa matriz. Para isto são utilizadas as entidades *hysteresis_sum_first*, *hysteresis_sum_second* e *hysteresis_sum_third*. Todas são compostas de instâncias da entidade *hysteresis_sum_element*, que é uma porta lógica "ou"com 16 entradas de 1 (um) bit e saída de 1 (um) bit. Internamente, as entidades *hysteresis_sum_first* e *hysteresis_sum_second* possuem, respectivamente, uma matriz de 16 x 16 e 4 x 4 instâncias de *hysteresis_sum_element*. No caso de *hysteresis_sum_third*, há apenas uma instância dessa entidade.

A idéia da implementação proposta é somar todos os elementos da matriz resultante de *hysteresis_check_pixel*, caso exista um ou mais pixels com vizinho eleito, retorne um valor

diferente de zero. Caso contrário, indica o término da execução da histerese para a imagem processada.

hysteresis_sum_first possui a entrada *bit_array_in* que é um barramento de 64 x 64 *bits*, e a saída *bit_array_out* que é um barramento de 16 x 16 *bits* (Figura 3.22). O código-fonte encontra-se no Anexo E.12.

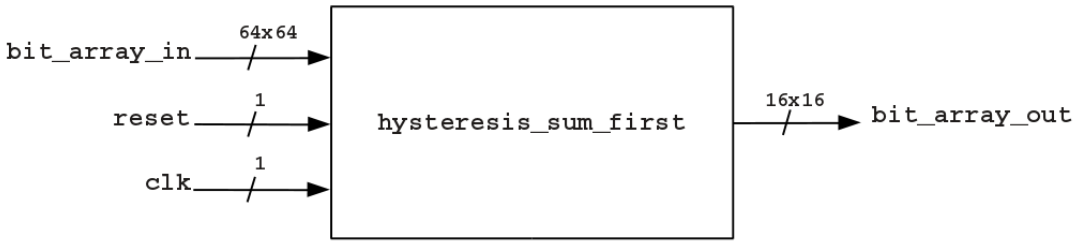

Figura 3.22: Interface de *hysteresis_sum_first* (Figura do autor)

hysteresis_sum_second possui a entrada *bit_array_in* que é um barramento de 16 x 16 *bits*, e a saída *bit_array_out* que é um barramento de 4 x 4 *bits* (Figura 3.23). O código-fonte encontra-se no Anexo E.13.

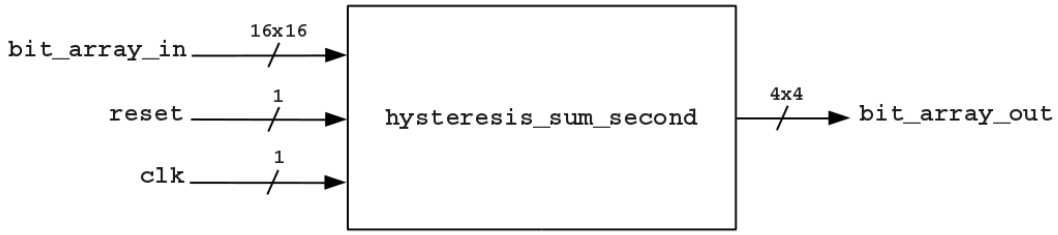

Figura 3.23: Interface de *hysteresis_sum_second* (Figura do autor)

hysteresis_sum_third possui a entrada *bit_array_in* que é um barramento de 4 x 4 *bits*, e a saída *bit_array_out* que é um sinal de um *bit* (Figura 3.24). O código-fonte encontra-se no Anexo E.14.

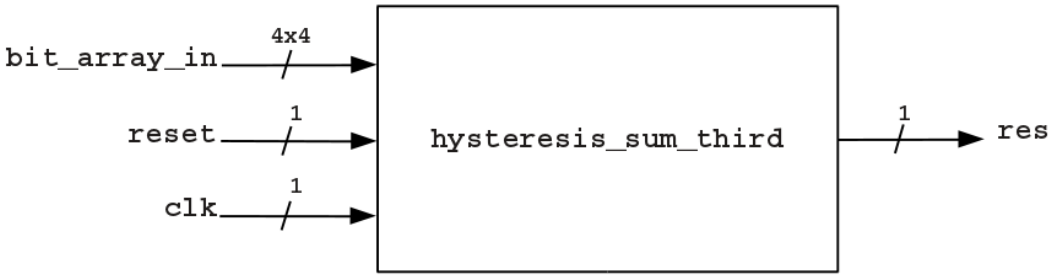

Figura 3.24: Interface de *hysteresis_sum_third* (Figura do autor)

Essas três entidades possuem em comum com a entidade *bubble_eval* apenas as entradas *reset* e *clk*.

A interface de *hysteresis_sum_element* é mostrada na Figura 3.25. As entradas *reset* e *clk* de *hysteresis_sum_element* são idênticas a *hysteresis_sum_first*. As entradas com prefixo *bit_* são os termos da soma e *bit_out* é o seu resultado. O código-fonte encontra-se no Anexo E.11.

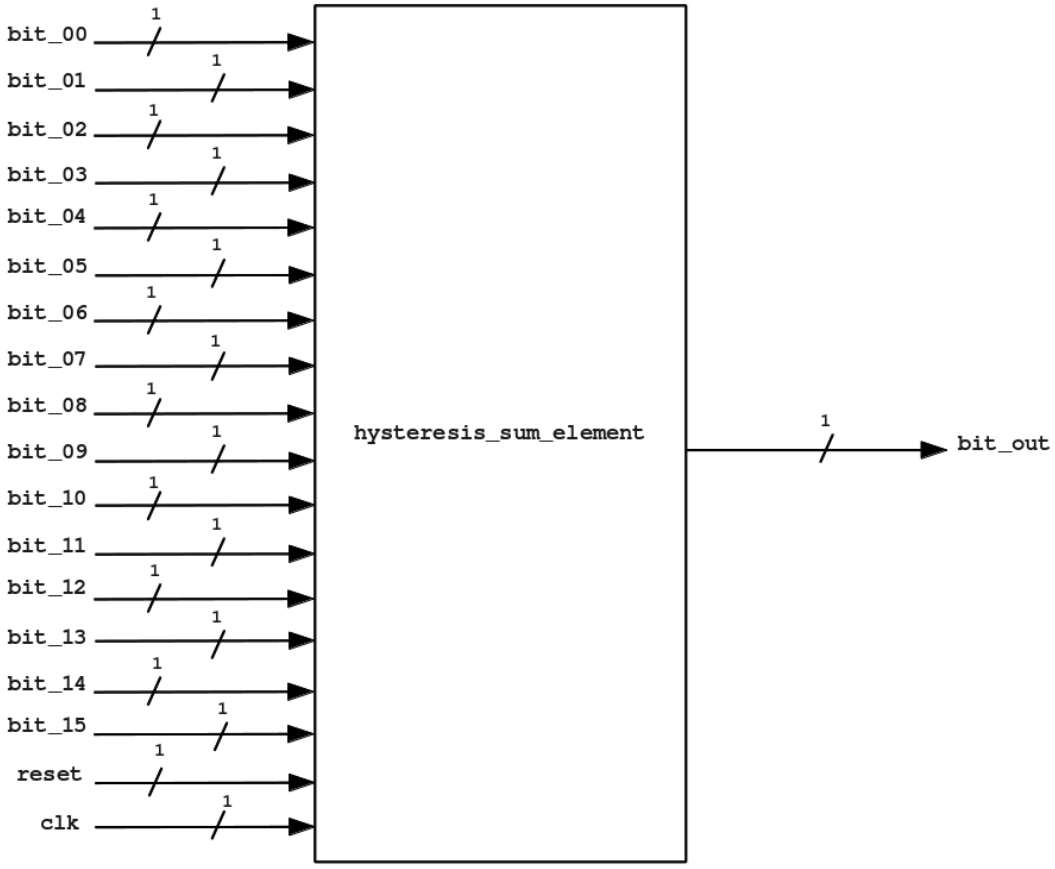

Figura 3.25: Interface de *hysteresis_sum_element* (Figura do autor)

O algoritmo 7, em pseudocódigo, descreve a funcionalidade de *hysteresis_sum_element*. O código-fonte em VHDL encontra-se no Anexo E.11.

Algoritmo 7 Algoritmo da entidade *hysteresis_sum_element*

*Resultado*1 ← *bit_*00 or *bit_*01 or *bit_*02 or *bit_*03
*Resultado*2 ← *bit_*04 or *bit_*05 or *bit_*06 or *bit_*07
*Resultado*3 ← *bit_*08 or *bit_*09 or *bit_*10 or *bit_*11
*Resultado*4 ← *bit_*12 or *bit_*13 or *bit_*14 or *bit_*15
bit_out ← *Resultado*1 or *Resultado*2 or *Resultado*3 or *Resultado*4

3.3.12 As entidades *hysteresis_out* e *hysteresis_out_element*

Nesta entidade é gerada imagem final nos quais os pixeis eleitos são transformados na cor branca e os pixeis restantes são transformados na cor preta. Internamente, *hysteresis_out* possui uma matriz com 64 x 64 instâncias de *hysteresis_out_element*, nas quais cada instância é responsável pela transformação num determinado pixel da imagem a ser processada.
Esta entidade possui todas as entradas e a saída semelhantes a *bubble*. A Figura 3.26 mostra a interface desta entidade. O código-fonte encontra-se no Anexo E.6.

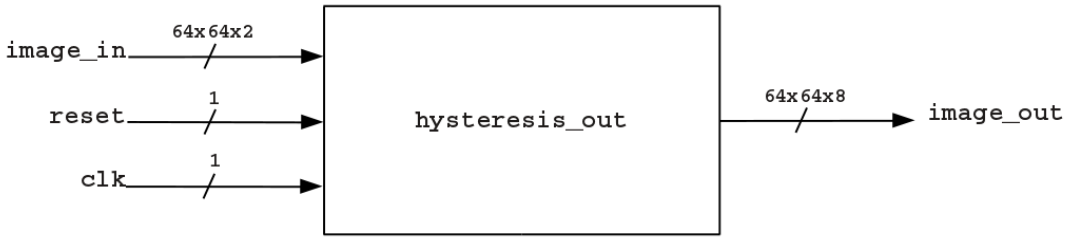

Figura 3.26: Interface de *hysteresis_out* (Figura do autor)

A interface de *hysteresis_out_element* está na Figura 3.27. As entradas *maxval*, *reset* e *clk* de *hysteresis_out_element* são idênticas a *hysteresis_out*. *pixel_in* é a intensidade da escala de cinza do pixel, *pixel_eval* é a sua qualificação como eleito, não eleito ou candidato a borda, e *pixel_out* é o seu resultado.

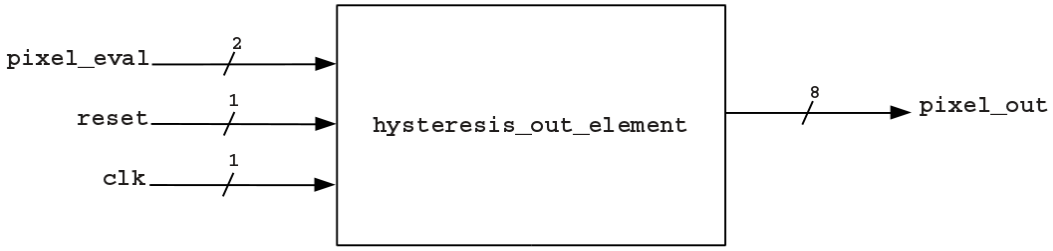

Figura 3.27: Interface de *hysteresis_out_element* (Figura do autor)

O algoritmo 8, em pseudocódigo, descreve a funcionalidade de *hysteresis_out_element*. O código-fonte em VHDL encontra-se no Anexo E.7.

Algoritmo 8 Algoritmo da entidade *hysteresis_out_element*

```
if pixel_eval = Eleito then
  pixel_out ← 255
else
  pixel_out ← 0
end if
```

CAPÍTULO 4

RESULTADOS E DISCUSSÕES

Este capítulo tem como objetivo mostrar os resultados obtidos da implementação proposta quanto ao espaço ocupado (Seção 4.1), quanto à qualidade da deteção de bordas comparando com a implementação do OpenCV (Seção 4.2) e quanto ao desempenho temporal da parte contida no FPGA(Seção 4.3).

4.1 Resultados referentes ao espaço ocupado

Quanto ao espaço ocupado pelo arquivo binário resultante da compilação, realizada com a otimização focada em área ocupada e no seu nível mais alto (nível 2), foram realizadas as seguintes análises:

- Algoritmo de Canny todo no FPGA que suporta imagens com 16x16 pixeis de 8 bits (Anexo A.1): O resultado foi um arquivo binário que utiliza 538% de *slices*, que é o agrupamento de 8 *Flip-flops* e 4 LUTs (*Look-up Tables*), no caso do FPGA Virtex-6. Logo, nota-se que o binário utiliza mais recursos do que o FPGA utilizado dispõe;

- Convolução Gaussiana que suporta imagens com 16x16 pixeis de 8 bits (Anexo A.2): O resultado foi um arquivo binário que utiliza 69% de *slices*;

- Histerese que suporta imagens com 16x16 pixeis de 8 bits, com *threshold* duplo e utilizando integer com limites (Anexo A.3): O resultado foi um arquivo binário que utiliza 34% de *slices*;

- Histerese que suporta imagens com 16x16 pixeis de 8 bits, com *threshold* duplo, com a preservação da instensidade e utilizando tipo *unsigned* (Anexo A.4): O resultado foi um arquivo binário que utiliza 19% de *slices*;

- Histerese que suporta imagens com 64x64 pixeis de 8 bits, com *threshold* duplo, com a preservação da instensidade e utilizando tipo *unsigned* (Anexo A.5): O resultado foi um arquivo binário que utiliza 140% de *slices* para LUT. Logo, nota-se que o binário utiliza mais recursos do que o FPGA utilizado dispõe;

- Histerese que suporta imagens com 16x16 pixeis de 8 bits, com *threshold* duplo, sem a preservação da instensidade e utilizando tipo *unsigned* (Anexo A.6): O resultado foi um arquivo binário que utiliza 13% de *slices*;

- Histerese que suporta matrizes com 16x16 rótulos de pixel (2 bits), sem *threshold* duplo, sem a preservação da instensidade e utilizando tipo *unsigned* (Anexo A.7): O resultado foi um arquivo binário que utiliza 6% de *slices*.

- Histerese que suporta matrizes com 64x64 rótulos de pixel (2 bits), sem *threshold* duplo, sem a preservação da instensidade e utilizando tipo *unsigned* (Anexo A.8): O resultado foi um arquivo binário que utiliza 92% de *slices*.

Pelo resultado das análises, é possível afirmar que não é possível alocar um projeto de circuito que processe a histerese com matrizes de rótulo de pixel maiores do que 64x64, no FPGA em questão, com a arquitetura adotada. Imagens maiores somente em um FPGA com espaço maior do que o presente no kit *Virtex-6 FPGA ML605* ou substituir a arquitetura adotada por uma arquitetura baseada em memórias.

4.2 Resultados referentes à qualidade de detecção das bordas

Para avaliar a qualidade de detecção foram utilizadas a base *"Berkeley Segmentation Dataset"* [38], que possui um conjunto de 100 (cem) imagens no formato JPEG (*Joint Photographic Experts Group*)[19], nas quais cada uma tem a resolução de 321 X 481 ou de 481 X 321 pixeis. Esta base também possui um outro conjunto de imagens que são a detecção de bordas realizadas por humanos das imagens de base.
Nesta avaliação, foram reduzidas as dimensões das imagens para 64 X 64 pixeis e convertidas para o formato PGM. Para realizar a conversão foi utilizado o script exposto no Anexo C.1.
As métricas utilizadas para avaliação são da Inês Boaventura e Adilson Gonzaga [6] que baseiam-se nas seguintes taxas:

- Taxa de pixeis de borda detectados corretamente (P_{co}), expressa na Equação 4.1, onde TP é a quantidade de pixeis de bordas corretamente detectadas, N_i é a quantidade de bordas de referência e N_b é a quantidade de bordas detectadas;

$$\boxed{P_{co} = \frac{TP}{max(N_i, N_b)}} \tag{4.1}$$

- Taxa de pixeis de borda não detectados (P_{nd}), expressa na Equação 4.2, onde FN é a quantidade de pixeis de bordas não detectadas, N_i é a quantidade de bordas de referência e N_b é a quantidade de bordas detectadas;

$$\boxed{P_{nd} = \frac{FN}{max(N_i, N_b)}} \tag{4.2}$$

- Taxa de pixeis que são falsos alarmes (P_{fa}), expressa na Equação 4.3, onde FP é a quantidade de pixeis que são falsos alarmes, ou seja, são pixeis que foram identificados como borda pelo algoritmo em avaliação, mas, não são bordas de acordo com o conjunto de referência. N_i é a quantidade de bordas de referência e N_b é a quantidade de bordas detectadas.

$$\boxed{P_{fa} = \frac{FP}{max(N_i, N_b)}} \tag{4.3}$$

O desvio padrão da máscara gaussiana é 1,4 e os limiares inferior e superior utilizados na histerese foram 50 e 150. Os valores foram escolhidos de forma empírica e foram utilizados tanto no projeto proposto quanto na implementação do OpenCV inalterada para uma avaliação justa e igualitária.
Para o cálculo das taxas citadas, foi utilizado o código implementado na linguagem C++ exposto no anexo B.2 em conjunto com o script exposto no anexo C.2.
A Tabela 4.1 mostra a média e o desvio padrão dos resultados da comparação do resultado ideal com a implementação proposta híbrida. A tabela com os resultados para cada imagem encontra-se no Anexo D.1.

	P_{co}	P_{nd}	P_{fa}
Média (%)	7,810094	21,943040	70,246976
Desvio padrão (%)	2,838041	12,591391	12,462853

Tabela 4.1: Tabela com os dados comparativos entre o resultado ideal e o resultado gerado pela implementação proposta híbrida

A Tabela 4.2 mostra a média e o desvio padrão dos resultados da comparação do resultado ideal com a implementação inalterada do OpenCV. A tabela com os resultados para cada imagem encontra-se no Anexo D.2.

	P_{co}	P_{nd}	P_{fa}
Média (%)	7,810103	21,943541	70,246356
Desvio padrão (%)	2,838086	12,591889	12,463371

Tabela 4.2: Tabela com os dados comparativos entre o resultado ideal e o resultado gerado pela implementação inalterada do OpenCV

A Tabela 4.3 mostra a média e o desvio padrão dos resultados da comparação da implementação inalterada do OpenCV com a implementação proposta híbrida. A tabela com os resultados para cada imagem encontra-se no Anexo D.3.

	P_{co}	P_{nd}	P_{fa}
Média (%)	99,997388	0	0,000261
Desvio padrão (%)	0,000412	0	0,000412

Tabela 4.3: Tabela com os dados comparativos entre o resultado ideal e o resultado gerado pela implementação inalterada do OpenCV

Com os resultados presentes nas Tabelas 4.1, 4.1 e 4.3 é possível observar que o algoritmo implementado em VHDL possui resultados muito similares ao da implementação presente na biblioteca OpenCV mesmo com abordagens de implementação diferentes.

4.3 Resultados referentes ao desempenho temporal

Nesta seção é abordado o desempenho temporal através da coleta da quantidade de ciclos gastos na parte implementada do FPGA. A tabela com a quantidade detalhada de ciclos por imagem encontra-se no Anexo D.4 e a Tabela 4.4 mostra a média, a mediana, o desvio padrão, o máximo e o mínimo dos dados do Anexo D.4.

	Quantidade de ciclos gastos
Média	465103,06
Mediana	438273,5
Desvio Padrão	196431,78
Mínimo	90113
Máximo	1101829

Tabela 4.4: Tabela com a média, a mediana, o desvio padrão, o mínimo e o máximo da quantidade de ciclos gastos pela parte do FPGA na implementação proposta

A frequência de um ciclo é inversamente proporcional ao seu período, como está expresso na Equação 4.4 na qual T é o período em segundos (s) e f é a frequência em Hertz (Hz) de acordo com o Sistema Internacional de Unidades [30].

$$\boxed{T = \frac{1}{f}} \tag{4.4}$$

Com a Equação 4.4 e a frequência utilizada de 100 MHz [27] foi possível calcular a média, a mediana, o desvio padrão, o mínimo e o máximo do tempo gasto que está presente na Tabela 4.5.

	Tempo gasto (milissegundos)
Média	4,6510306
Mediana	4,382735
Desvio Padrão	1,9643178
Mínimo	0,90113
Máximo	11,01829

Tabela 4.5: Tabela com a média, a mediana, o desvio padrão, o mínimo e o máximo do tempo gasto pela parte do FPGA na implementação proposta

Para comparação, foi medido o tempo gasto da histerese da biblioteca OpenCV. O código encontra-se no Anexo B.3 e, na Tabela 4.6, encontra-se a média, a mediana, o desvio padrão, o mínimo e o máximo do tempo gasto. No Anexo D.5 encontra-se o tempo gasto pela histerese do OpenCV para cada imagem da base adotada.

	Tempo gasto (milissegundos)
Média	0,01885053
Mediana	0,0188185
Desvio Padrão	0,00579246
Mínimo	0,007274
Máximo	0,043523

Tabela 4.6: Tabela com a média, a mediana, o desvio padrão, o mínimo e o máximo do tempo gasto pela histerese da biblioteca OpenCV

Comparando os dados das tabelas 4.5 e 4.6, pode-se notar que o tempo da implementação do OpenCV foi melhor do que a da implementação proposta. Todavia, boa parte do tempo gasto da implementação proposta é resultante da transferência de dados (entrada e saída dos rótulos de pixeis) e do processo de conversão dos dados, de serial para paralelo e paralelo para serial. Nos casos de transferência de dados e conversão, estes levam 4096 (64 x 64) cíclos para cada imagem que entra e 4096 cíclos para cada imagem que sai. Logo, pode-se concluir que gasta-se 8192 cíclos somente para entrada e saída da imagem.
Outro fator acarreta custo computacional na implementação proposta é o seu controle, na qual o sinal que indica que há uma imagem inteira carregada da entidade *serial_to_parallel*, o sinal *ready*, dita a sincronia de *hysteresis_pipeline*, gerando um controle mestre-escravo. Logo, a cada 4096 cíclos, quando há uma imagem carregada, os dados passam de um estágio para o outro no *pipeline* interno a entidade *hysteresis_pipeline*.
Com a retirada dos cíclos gastos da transferência de dados e considerando que cada estágio do *pipeline* interno a entidade *hysteresis_pipeline* leve um cíclo de execução, é possível calcular a quantidade de cíclos líquidos, que são os cíclos gastos apenas no

processamento dos rótulos de imagem, cuja a Equação 4.5 expressa o como é realizado este cálculo. CL e CG representam a quantidade de cíclos líquidos e o total de ciclos gastos, respectivamente.

$$\boxed{CL = \frac{CG - 8192}{4096}} \tag{4.5}$$

Com a Equação 4.5 foi possível encontrar a quantidade de cíclos líquidos para cada imagem presente no Anexo D.6 e, na Tabela 4.7, encontra-se a média, a mediana, o desvio padrão, o mínimo e o máximo da quantidade de ciclos gastos.

	Quantidade de estágios realizados
Média	111,55
Mediana	105
Desvio Padrão	47,96
Mínimo	20
Máximo	267

Tabela 4.7: Tabela com a média, a mediana, o desvio padrão, o mínimo e o máximo da quantidade de cíclos líquidos na implementação proposta

Com os dados presentes no Anexo D.6 e a frequência utilizada de 100 MHz [27] foi possível estimar o tempo líquido para cada imagem, presente no Anexo D.7. Na Tabela 4.8, encontra-se a média, a mediana, o desvio padrão, o mínimo e o máximo do tempo líquido estimado na implementação proposta.

	Tempo estimado (milissegundos)
Média	0,0011155
Mediana	0,0010500
Desvio Padrão	0,0004796
Mínimo	0,0002000
Máximo	0,0026700

Tabela 4.8: Tabela com a média, a mediana, o desvio padrão, o mínimo e o máximo do tempo líquido estimado na implementação proposta

Comparando os dados da Tabela 4.8 com o resultado da implementação do OpenCV, presente na Tabela 4.6, pode-se notar que o tempo líquido da implementação proposta foi superior ao tempo da implementação do OpenCV.

CAPÍTULO 5

CONCLUSÃO

Neste trabalho foi apresentado um projeto híbrido, parte no PC e parte no FPGA Xilinx Virtex 6, do algoritmo de Canny com uma arquitetura que explora a arquitetura em *pipeline* e o paralelismo entre os pixeis (Anexo A.8). Na parte do PC, foi utilizado parte do código-fonte da função *Canny* da biblioteca OpenCV, e, no FPGA foi utilizada a linguagem de descrição VHDL. O ponto interessante deste trabalho foi o projeto que combina as arquiteturas em *pipeline* e o paralelismo entre os pixeis, algo que não foi explorado em outros projetos e implementações, como pôde ser visto na Seção 2.5.
As desvantagens foram as seguintes:

- A contraprodutividade no desenvolvimento do projeto: devido a linguagem de descrição VHDL e a tecnologia utilizada, no caso o FPGA, levou-se um tempo maior para desenvolver este projeto do que se fosse desenvolvido apenas no PC utilizando uma outra linguagem de programação imperativa;
- O grande consumo de espaço do FPGA: devido ao caráter da arquitetura proposta, é utilizado uma quantidade grande de *slices* do FPGA (Seção 4.1), muito maior do que arquiteturas que usam memória, por exemplo [33]. Consequentemente, este foi um fator que limitou o uso de imagens maiores do que as que foram testadas. Como também limitou uma implementação completa do algoritmo de Canny apenas no FPGA.

Quanto ao circuito que computa a histerese no FPGA, este demonstrou ser similar ao que foi implementado no OpenCV, no que se refere aos resultados demonstrados na Seção 4.2. Pois, ambas são bem distintas, a primeira tem como enfoque apenas a transformação dos pixels candidatos em eleitos, dado a sua vizinhança, e, a segunda percorre todos os pixels eleitos e depois faz a expansão das bordas transformando os candidatos em eleitos.

5.1 Trabalho Futuro

Como trabalho futuro, fica a inserção da biblioteca OpenCV no *Memory Endpoint Test Driver* (MET) [4]. O MET é um conjunto de aplicativos que permitem a comunicação entre o FPGA e o computador hospedeiro usando o *PCI-Express* com o método de tranferência PIO. No MET há o *driver* do dispositivo binário para sitemas Microsoft

Windows, o código-fonte do *driver* do dispostivo para Linux (o autor testou no Linux Fedora 10) e o código-fonte para o aplicativo que escreve e lê os dados no *driver*. Foram necessárias alterações no código-fonte do *driver* para que o driver fosse compatível com versões mais recentes do *kernel* Linux e para remover um erro que impedia enviar mais do que 7 (sete) *bytes* para o FPGA em uma só vez. Na documentação constava que poderia ser enviado até 8192 (oito mil cento e noventa e dois) *bytes* de uma só vez. No anexo B.4, consta este código-fonte modificado.

BIBLIOGRAFIA

[1] Accame, Marco e Natale, Francesco G. B. De. Edge detection by point classification of canny filtered images. *Signal Processing*, 60(1):11–22, julho de 1997.

[2] Amaricai, A., Boncalo, O., Iordate, M., e Marinescu, B. A moving window architecture for a hw/sw codesign based canny edge detection for fpga. *Microelectronics (MIEL), 2012 28th International Conference on*, páginas 393–396, 2012.

[3] Peter J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kauffman Publishers, third edition, 2008.

[4] Ayer Jr., John. Using the memory endpoint test driver (met) with the programmed input/output example design for pci express endpoint cores. Disponível em: <http://www.xilinx.com/support/documentation/application_-notes/xapp1022.pdf> . Acesso em: 9 de setembro de 2014.

[5] BDTI. Synfora's pico high-level synthesis tool achieves bdti certification. Disponível em: <http://www.bdti.com/InsideDSP/2010/03/18/Synfora> . Acesso em: 11 de setembro de 2013.

[6] Boaventura, Inês A. G. e Gonzaga, Adilson. Método de avaliação de detector de bordas em imagens digitais. 2009.

[7] Bradski, Gary. The opencv library. *Dr. Dobb's Journal of Software Tools*, 25(11):120–126, 2000.

[8] Canny, John F. Finding edges and lines in images. Dissertação de Mestrado, Cambridge, MA, USA, 1983.

[9] Canny, John F. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis an Machine Intelligence*, PAMI-8(6):679–714, novembro de 1986.

[10] Chaitra, N. M. e Ramana Reddy, K. V. Implementation of canny edge detection algorithm on fpga and displaying image through vga interface. *International Journal of Engineering and Advanced Technology (IJEAT)*, 2(6):243–247, agosto de 2013.

[11] Chu, Pong P. *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. Wiley, 2008.

[12] Altera Corporation. Altera quartus ii. Disponível em: <http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html> . Acesso em: 11 de setembro de 2013.

[13] Lattice Semiconductor Corporation. Lattice diamond. Disponível em: <http://www.latticesemi.com/LatticeDiamond> . Acesso em: 11 de setembro de 2013.

[14] Microsemi Corporation. Libero soc. Disponível em: <http://www.microsemi.com/products/fpga-soc/design-resources/design-software/libero-soc>. Acesso em: 11 de setembro de 2013.

[15] Desmouliers, C., Aslan, S., Oruklu, E., Saniie, J., e Vallina, F.M. Hw/sw co-design platform for image and video processing applications on virtex-5 fpga using pico. *Electro/Information Technology (EIT), 2010 IEEE International Conference on*, páginas 1–6, 2010.

[16] Do Vale, Giovane Maia e Dal Poz, Aluir Porfírio. Processo de detecção de bordas de canny. *Boletim de Ciências Geodésicas*, 8(2), 2004.

[17] Gentsos, Christos, Sotiropoulou, Calliope-Louisa, Nikolaidis, Spiridon, e Vassiliadis, Nikolaos. Real-time canny edge detection parallel implementation for fpgas. *ICECS*, páginas 499–502. IEEE, 2010.

[18] Gibson, R. M., McMeekin, S.G., Ahmadinia, A., Strang, N. C., e Morison, G. Evaluation of visual aid enhancement algorithms for real-time embedded systems. *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, páginas 1762–1769, 2012.

[19] Joint Photographic Experts Group. Jpeg homepage. Disponível em: <http://www.jpeg.org/jpeg/index.html>. Acesso em: 9 de setembro de 2014.

[20] PCI Special Interest Group. Pci-sig pci express. Disponível em: <http://www.pcisig.com/specifications/pciexpress> . Acesso em: 11 de setembro de 2013.

[21] Haghi, A., Sheikh, U.U., e Marsono, M.N. A hardware/software co-design architecture of canny edge detection. *Computational Intelligence, Modelling and Simulation (CIMSiM), 2012 Fourth International Conference on*, páginas 214–219, 2012.

[22] He, Wenhao e Yuan, Kui. An improved canny edge detector and its realization on fpga. *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, páginas 6561–6564, 2008.

[23] Henderson, Bryan. Netpbm history, fevereiro de 2007. Disponível em: <http://netpbm.sourceforge.net/history.html> . Acesso em: 11 de setembro de 2013.

[24] Houari, Kobzili El, Cherrad, Benbouchama, e Zohir, Irki. A software-hardware mixed design for the fpga implementation of the real-time edge detection. *SMC*, páginas 4091–4095. IEEE, 2010.

[25] National Instruments Inc. Introdução à tecnologia fpga, dezembro de 2011. Disponível em: <http://www.ni.com/white-paper/6984/pt/> . Acesso em: 11 de setembro de 2013.

[26] Xilinx Inc. Ise design suite. Disponível em: <http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm> . Acesso em: 11 de setembro de 2013.

[27] Xilinx Inc. Virtex-6 fpga integrated block for pci express: User guide. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/v6_-pcie/v2_5/ug671_V6_IntBlock_PCIe.pdf> . Acesso em: 11 de setembro de 2013.

[28] Xilinx Inc. Virtex-6 fpga ml605 evaluation kit. Disponível em: <http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm> . Acesso em: 11 de setembro de 2013.

[29] Xilinx Inc. Virtex-6 family overview, janeiro de 2012. Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf> . Acesso em: 11 de setembro de 2013.

[30] International Bureau of Weights and Measures, Taylor, Barry N., e Thompson, Ambler. *The international system of units (SI)*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2001.

[31] Jain, Ramesh, Kasturi, Rangachar, e Schunck, Brian G. *Machine vision*. McGraw-Hill, Inc., New York, NY, USA, 1995.

[32] Jeff Johnson. List and comparison of fpga companies, julho de 2011. Disponível em: <http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html>. Acesso em: 11 de setembro de 2013.

[33] Kelefouras, Vasilios, Kritikakou, Angeliki, e Goutis, Costas. A methodology for speeding up edge and line detection algorithms focusing on memory architecture utilization. *The Journal of Supercomputing*, 68(1):459–487, 2014.

[34] Li, Xiaoyang, Jiang, Jie, e Fan, Qiaoyun. An improved real-time hardware architecture for canny edge detection based on fpga. *Intelligent Control and Information Processing (ICICIP), 2012 Third International Conference on*, páginas 445–449, 2012.

[35] Lorca, F. G., Kessal, Lounis, e Demigny, Didier. Efficient asic and fpga implementations of iir filters for real time edge detection. *ICIP (2)*, páginas 406–409, 1997.

[36] Lourenço, Luis Henrique Alves. Paralelização do detector de bordas canny para a biblioteca itk utilizando cuda. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, PR, Brasil, 2011.

[37] Luk, Wayne, Wu, Teddy, e Page,Ian. Hardware-software codesign of multidimensional programs. *in Proc. FCCM94, D. Buell and K.L. Pocek (eds.), IEEE Computer*, páginas 82–90. Society Press, 1994.

[38] Martin, David R., Fowlkes, Charless, Tal, Doron, e Malik, Jitendra. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *Proc. 8th Int'l Conf. Computer Vision*, volume 2, páginas 416–423, julho de 2001.

[39] Medina Carnicer, Rafael, Muñoz-Salinas, Rafael, Yeguas-Bolivar, Enrique, e Díaz-Más, L. A novel method to look for the hysteresis thresholds for the canny edge detector. *Pattern Recognition*, 44(6):1201–1211, 2011.

[40] Muthukumar, V. e Rao, Daggu Venkateshwar. Image processing algorithms on reconfigurable architecture using handelc. *2012 15th Euromicro Conference on Digital System Design*, 0:218–226, 2012.

[41] Nadernejad, Ehsan, Sharifzadeh, Sara, e Hassanpour, Hamid. Edge detection techniques: Evaluations and comparison. *Applied Mathematical Sciences*, 2(31):1507–1520, 2008.

[42] Neoh, Hong Shan e Hazanchuk, Asher. Adaptive edge detection for real-time video processing using fpgas, 2005.

[43] Pankiewicz, P., Powiertowski, W., e Roszak, G. Vhdl implementation of the lane detection algorithm. *Mixed Design of Integrated Circuits and Systems, 2008. MIXDES 2008. 15th International Conference on*, páginas 581–584, 2008.

[44] Peng, Fangxin, Lu, Xiaofeng, Lu, Hengli, e Shen, Sumin. An improved high-speed canny edge detection algorithm and its implementation on fpga. volume 8350, páginas 83501V–83501V–7, 2011.

[45] Poskanzer, Jef. Netpbm format specification, outubro de 2003. Disponível em: <http://netpbm.sourceforge.net/doc/pgm.html> . Acesso em: 11 de setembro de 2013.

[46] Prewitt, J. M. S. Object enhancement and extraction. *Proceedings of The IEEE*, 1970.

[47] Pridmore, Tony P. Thresholding images of line drawings with hysteresis. Blostein, Dorothea e Kwon, Young-Bin, editors, *GREC*, volume 2390 of *Lecture Notes in Computer Science*, páginas 310–319. Springer, 2001.

[48] Roberts, Lawrence G. Machine perception of three-dimensional solids. *MIT Lincoln Laboratory Technical Report*, (315), maio de 1963.

[49] Scharr, H., Körkel, S., e Jähne, B. Numerische isotropieoptimierung von fir-filtern mittels querglättung. páginas 367–374, 1997.

[50] Sen,Debashis e Pal, Sankar K. Gradient histogram: Thresholding in a region of interest for edge detection. *Image Vision Comput.*, 28(4):677–695, 2010.

[51] Sobel, Irwin e Feldman, Gary. A 3x3 isotropic gradient operator for image processing. 1968.

[52] Stroustrup, Bjarne. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 1997.

[53] Trost, Andrej, Zemva, Andrej, e Zajc, Baldomir. Educational programmable hardware prototyping and verification system. *Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, FPL '00, páginas 818–821, London, UK, 2000. Springer-Verlag.

[54] Xu,Qian, Chakrabarti, C., e Karam, L.J. A distributed canny edge detector and its implementation on fpga. *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011 IEEE*, páginas 500–505, 2011.

[55] Xu,Qian, Varadarajan, S., Chakrabarti, C., e Karam, L.J. A distributed canny edge detector: Algorithm and fpga implementation. *IEEE Transactions on Image Processing*, 23(7):2944–2960, julho de 2014.

[56] Zhang, Yang e Rockett, Peter. The bayesian operating point of the canny edge detector. *IEEE Transactions on Image Processing*, 15(11):3409–3416, 2006.

ANEXO A

RELATÓRIOS DE COMPILAÇÃO (*LOGS*)

A.1 Relatório de compilação do Canny completo que suporta imagens com 16x16 *pixels*

```
Device utilization summary:
---------------------------

Selected Device : 6vlx240tff1156-1


Slice Logic Utilization:
 Number of Slice Registers:           55142  out of  301440    18%
 Number of Slice LUTs:                812141  out of  150720   538% (*)
    Number used as Logic:             812136  out of  150720   538% (*)
    Number used as Memory:                 5  out of  58400     0%
       Number used as SRL:                 5

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  837032
   Number with an unused Flip Flop:   781890  out of  837032    93%
   Number with an unused LUT:         24891  out of  837032     2%
   Number of fully used LUT-FF pairs: 30251  out of  837032     3%
   Number of unique control sets:      6296

IO Utilization:
 Number of IOs:                          19
 Number of bonded IOBs:                  19  out of    600     3%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                8  out of    416     1%
    Number using Block RAM only:          8
 Number of BUFG/BUFGCTRLs:                5  out of     32    15%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used
```

A.2 Relatório de compilação do Gaussiano que suporta imagens com 16x16 *pixels*

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                46,983 out of 301,440   15%
    Number used as Flip Flops:              15,482
    Number used as Latches:                      1
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:           31,500
  Number of Slice LUTs:                     84,166 out of 150,720   55%
    Number used as logic:                   82,415 out of 150,720   54%
      Number using O6 output only:          33,524
      Number using O5 output only:           8,948
      Number using O5 and O6:               39,943
      Number used as ROM:                        0
    Number used as Memory:                       5 out of  58,400    1%
      Number used as Dual Port RAM:              0
      Number used as Single Port RAM:            0
      Number used as Shift Register:             5
        Number using O6 output only:             5
        Number using O5 output only:             0
        Number using O5 and O6:                  0
    Number used exclusively as route-thrus:  1,746
      Number with same-slice register load:  1,222
      Number with same-slice carry load:       524
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                26,103 out of  37,680   69%
  Number of LUT Flip Flop pairs used:       86,258
    Number with an unused Flip Flop:        42,633 out of  86,258   49%
    Number with an unused LUT:               2,092 out of  86,258    2%
    Number of fully used LUT-FF pairs:      41,533 out of  86,258   48%
    Number of slice register sites lost
      to control set restrictions:               0 out of 301,440    0%
```

A.3 Relatório de compilação da Histerese que suporta imagens com 16x16 *pixels*, com *threshold* duplo e utilizando integer com limites (*range*)

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                31,844 out of 301,440   10%
    Number used as Flip Flops:              27,943
    Number used as Latches:                      1
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:            3,900
  Number of Slice LUTs:                     28,482 out of 150,720   18%
    Number used as logic:                   25,870 out of 150,720   17%
      Number using O6 output only:          16,015
      Number using O5 output only:           1,019
      Number using O5 and O6:                8,836
      Number used as ROM:                        0
    Number used as Memory:                       5 out of  58,400    1%
      Number used as Dual Port RAM:              0
      Number used as Single Port RAM:            0
      Number used as Shift Register:             5
        Number using O6 output only:             5
        Number using O5 output only:             0
        Number using O5 and O6:                  0
    Number used exclusively as route-thrus:  2,607
      Number with same-slice register load:  1,749
      Number with same-slice carry load:       858
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                13,058 out of  37,680   34%
  Number of LUT Flip Flop pairs used:       38,977
    Number with an unused Flip Flop:        11,072 out of  38,977   28%
    Number with an unused LUT:              10,495 out of  38,977   26%
    Number of fully used LUT-FF pairs:      17,410 out of  38,977   44%
    Number of unique control sets:           1,397
    Number of slice register sites lost
```

```
    to control set restrictions:          4,763 out of 301,440    1%
```

A.4 Relatório de compilação da Histerese que suporta imagens com 16x16 *pixels*, com *threshold* duplo, com a preservação da instensidade e utilizando tipo *unsigned*

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:               20,768 out of 301,440    6%
    Number used as Flip Flops:             20,767
    Number used as Latches:                     1
    Number used as Latch-thrus:                 0
    Number used as AND/OR logics:               0
  Number of Slice LUTs:                    16,557 out of 150,720   10%
    Number used as logic:                  14,403 out of 150,720    9%
      Number using O6 output only:         11,829
      Number using O5 output only:            722
      Number using O5 and O6:                1,852
      Number used as ROM:                        0
    Number used as Memory:                       5 out of  58,400    1%
      Number used as Dual Port RAM:              0
      Number used as Single Port RAM:            0
      Number used as Shift Register:             5
        Number using O6 output only:             5
        Number using O5 output only:             0
        Number using O5 and O6:                  0
    Number used exclusively as route-thrus:  2,149
      Number with same-slice register load:  1,610
      Number with same-slice carry load:       539
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                7,451 out of  37,680   19%
  Number of LUT Flip Flop pairs used:      22,108
    Number with an unused Flip Flop:        4,102 out of  22,108   18%
    Number with an unused LUT:              5,551 out of  22,108   25%
    Number of fully used LUT-FF pairs:     12,455 out of  22,108   56%
```

```
    Number of slice register sites lost
      to control set restrictions:              0 out of 301,440    0%
```

A.5 Relatório de compilação da Histerese que suporta imagens com 64x64 *pixels*, com *threshold* duplo, com a preservação da instensidade e utilizando tipo *unsigned*

```
Device utilization summary:
---------------------------

Selected Device : 6vlx240tff1156-1


Slice Logic Utilization:
 Number of Slice Registers:           361558  out of  301440   119% (*)
 Number of Slice LUTs:                211363  out of  150720   140% (*)
    Number used as Logic:             211358  out of  150720   140% (*)
    Number used as Memory:                 5  out of  58400     0%
       Number used as SRL:                 5

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  572921
   Number with an unused Flip Flop:   211363  out of  572921    36%
   Number with an unused LUT:         361558  out of  572921    63%
   Number of fully used LUT-FF pairs:      0  out of  572921     0%
   Number of unique control sets:      54587

IO Utilization:
 Number of IOs:                           19
 Number of bonded IOBs:                   19  out of    600     3%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                 8  out of    416     1%
    Number using Block RAM only:           8
 Number of BUFG/BUFGCTRLs:                 5  out of     32    15%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used
```

A.6 Relatório de compilação da Histerese que suporta imagens com 16x16 *pixels*, com *threshold* duplo, sem a preservação da instensidade e utilizando tipo *unsigned*

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                10,016 out of 301,440    3%
    Number used as Flip Flops:              10,015
    Number used as Latches:                      1
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                     11,028 out of 150,720    7%
    Number used as logic:                   10,298 out of 150,720    6%
      Number using O6 output only:           9,282
      Number using O5 output only:             444
      Number using O5 and O6:                  572
      Number used as ROM:                        0
    Number used as Memory:                       5 out of  58,400    1%
      Number used as Dual Port RAM:              0
      Number used as Single Port RAM:            0
      Number used as Shift Register:             5
        Number using O6 output only:             5
        Number using O5 output only:             0
        Number using O5 and O6:                  0
    Number used exclusively as route-thrus:    725
      Number with same-slice register load:    442
      Number with same-slice carry load:       283
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                 4,924 out of  37,680   13%
  Number of LUT Flip Flop pairs used:       12,730
    Number with an unused Flip Flop:         3,405 out of  12,730   26%
    Number with an unused LUT:               1,702 out of  12,730   13%
    Number of fully used LUT-FF pairs:       7,623 out of  12,730   59%
    Number of slice register sites lost
      to control set restrictions:               0 out of 301,440    0%
```

A.7 Relatório de compilação da Histerese que suporta imagens com 16x16 *pixels*, sem *threshold* duplo, sem a preservação da instensidade e utilizando tipo *unsigned*

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                 6,151 out of 301,440    2%
    Number used as Flip Flops:               6,150
    Number used as Latches:                      1
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                      5,219 out of 150,720    3%
    Number used as logic:                    4,307 out of 150,720    2%
      Number using O6 output only:           3,318
      Number using O5 output only:             149
      Number using O5 and O6:                  840
      Number used as ROM:                        0
    Number used as Memory:                       5 out of  58,400    1%
      Number used as Dual Port RAM:              0
      Number used as Single Port RAM:            0
      Number used as Shift Register:             5
        Number using O6 output only:             5
        Number using O5 output only:             0
        Number using O5 and O6:                  0
    Number used exclusively as route-thrus:    907
      Number with same-slice register load:    624
      Number with same-slice carry load:       283
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                 2,610 out of  37,680    6%
  Number of LUT Flip Flop pairs used:        6,696
    Number with an unused Flip Flop:         1,840 out of   6,696   27%
    Number with an unused LUT:               1,477 out of   6,696   22%
    Number of fully used LUT-FF pairs:       3,379 out of   6,696   50%
    Number of slice register sites lost
      to control set restrictions:               0 out of 301,440    0%
```

A.8 Relatório de compilação da Histerese que suporta imagens com 64x64 *pixels*, sem *threshold* duplo, sem a preservação da instensidade e utilizando tipo *unsigned*

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                81,150 out of 301,440   26%
    Number used as Flip Flops:              81,149
    Number used as Latches:                      1
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                     62,564 out of 150,720   41%
    Number used as logic:                   55,390 out of 150,720   36%
      Number using O6 output only:          40,504
      Number using O5 output only:           6,232
      Number using O5 and O6:                8,654
      Number used as ROM:                        0
    Number used as Memory:                     13 out of  58,400    1%
      Number used as Dual Port RAM:                0
      Number used as Single Port RAM:              0
      Number used as Shift Register:              13
        Number using O6 output only:              13
        Number using O5 output only:               0
        Number using O5 and O6:                    0
    Number used exclusively as route-thrus:  7,161
      Number with same-slice register load:  3,038
      Number with same-slice carry load:     4,123
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                34,710 out of  37,680   92%
  Number of LUT Flip Flop pairs used:       92,999
    Number with an unused Flip Flop:        20,703 out of  92,999   22%
    Number with an unused LUT:              30,435 out of  92,999   32%
    Number of fully used LUT-FF pairs:      41,861 out of  92,999   45%
    Number of slice register sites lost
```

```
to control set restrictions:                0 out of 301,440    0%
```

ANEXO B

TRECHOS DE CÓDIGO-FONTE

B.1 Função do algoritmo de Canny sem a segunda fase da histerese (baseado no código-fonte da biblioteca OpenCV)

```
1  void almost_Canny( InputArray _src, OutputArray _dst,
2                     double low_thresh, double high_thresh,
3                     int aperture_size, bool L2gradient )
4  {
5      Mat src = _src.getMat();
6      CV_Assert( src.depth() == CV_8U );
7
8      _dst.create(src.size(), CV_8U);
9      Mat dst = _dst.getMat();
10
11     if (!L2gradient && (aperture_size & CV_CANNY_L2_GRADIENT) ==
           CV_CANNY_L2_GRADIENT)
12     {
13         //backward compatibility
14         aperture_size &= ~CV_CANNY_L2_GRADIENT;
15         L2gradient = true;
16     }
17
18     if ((aperture_size & 1) == 0 || (aperture_size != -1 && (aperture_size
           < 3 || aperture_size > 7)))
19         CV_Error(CV_StsBadFlag, "");
20
21     if (low_thresh > high_thresh)
22         std::swap(low_thresh, high_thresh);
23
24 #ifdef HAVE_TEGRA_OPTIMIZATION
25     if (tegra::canny(src, dst, low_thresh, high_thresh, aperture_size,
           L2gradient))
26         return;
27 #endif
28
29 #ifdef USE_IPP_CANNY
30     if( aperture_size == 3 && !L2gradient &&
31         ippCanny(src, dst, (float)low_thresh, (float)high_thresh) )
32         return;
33 #endif
```

```
34
35      const int cn = src.channels();
36      Mat dx(src.rows, src.cols, CV_16SC(cn));
37      Mat dy(src.rows, src.cols, CV_16SC(cn));
38
39      Sobel(src, dx, CV_16S, 1, 0, aperture_size, 1, 0, cv::BORDER_REPLICATE)
            ;
40      Sobel(src, dy, CV_16S, 0, 1, aperture_size, 1, 0, cv::BORDER_REPLICATE)
            ;
41
42      if (L2gradient)
43      {
44          low_thresh = std::min(32767.0, low_thresh);
45          high_thresh = std::min(32767.0, high_thresh);
46
47          if (low_thresh > 0) low_thresh *= low_thresh;
48          if (high_thresh > 0) high_thresh *= high_thresh;
49      }
50      int low = cvFloor(low_thresh);
51      int high = cvFloor(high_thresh);
52
53      ptrdiff_t mapstep = src.cols + 2;
54      AutoBuffer<uchar> buffer((src.cols+2)*(src.rows+2) + cn * mapstep * 3 *
            sizeof(int));
55
56      int* mag_buf[3];
57      mag_buf[0] = (int*)(uchar*)buffer;
58      mag_buf[1] = mag_buf[0] + mapstep*cn;
59      mag_buf[2] = mag_buf[1] + mapstep*cn;
60      memset(mag_buf[0], 0, /* cn* */mapstep*sizeof(int));
61
62      uchar* map = (uchar*)(mag_buf[2] + mapstep*cn);
63      memset(map, 1, mapstep);
64      memset(map + mapstep*(src.rows + 1), 1, mapstep);
65
66      int maxsize = std::max(1 << 10, src.cols * src.rows / 10);
67      std::vector<uchar*> stack(maxsize);
68      uchar **stack_top = &stack[0];
69      uchar **stack_bottom = &stack[0];
70
71      /* sector numbers
72         (Top-Left Origin)
73
74          1   2   3
75           *  *  *
76            * * *
77          0*******0
```

```
78              * * *
79             *   *   *
80            3    2    1
81        */
82
83        #define CANNY_PUSH(d)     *(d) = uchar(2), *stack_top++ = (d)
84        #define CANNY_POP(d)      (d) = *--stack_top
85
86        // calculate magnitude and angle of gradient, perform non-maxima
              supression.
87        // fill the map with one of the following values:
88        //   0 - the pixel might belong to an edge
89        //   1 - the pixel can not belong to an edge
90        //   2 - the pixel does belong to an edge
91        for (int i = 0; i <= src.rows; i++)
92        {
93            int* _norm = mag_buf[(i > 0) + 1] + 1;
94            if (i < src.rows)
95            {
96                short* _dx = dx.ptr<short>(i);
97                short* _dy = dy.ptr<short>(i);
98
99                if (!L2gradient)
100               {
101                   for (int j = 0; j < src.cols*cn; j++)
102                       _norm[j] = std::abs(int(_dx[j])) + std::abs(int(_dy[j])
                              );
103               }
104               else
105               {
106                   for (int j = 0; j < src.cols*cn; j++)
107                       _norm[j] = int(_dx[j])*_dx[j] + int(_dy[j])*_dy[j];
108               }
109
110               if (cn > 1)
111               {
112                   for(int j = 0, jn = 0; j < src.cols; ++j, jn += cn)
113                   {
114                       int maxIdx = jn;
115                       for(int k = 1; k < cn; ++k)
116                           if(_norm[jn + k] > _norm[maxIdx]) maxIdx = jn + k;
117                       _norm[j] = _norm[maxIdx];
118                       _dx[j] = _dx[maxIdx];
119                       _dy[j] = _dy[maxIdx];
120                   }
121               }
122               _norm[-1] = _norm[src.cols] = 0;
```

```
123         }
124         else
125             memset(_norm-1, 0, /* cn* */mapstep*sizeof(int));
126
127         // at the very beginning we do not have a complete ring
128         // buffer of 3 magnitude rows for non-maxima suppression
129         if (i == 0)
130             continue;
131
132         uchar* _map = map + mapstep*i + 1;
133         _map[-1] = _map[src.cols] = 1;
134
135         int* _mag = mag_buf[1] + 1; // take the central row
136         ptrdiff_t magstep1 = mag_buf[2] - mag_buf[1];
137         ptrdiff_t magstep2 = mag_buf[0] - mag_buf[1];
138
139         const short* _x = dx.ptr<short>(i-1);
140         const short* _y = dy.ptr<short>(i-1);
141
142         if ((stack_top - stack_bottom) + src.cols > maxsize)
143         {
144             int sz = (int)(stack_top - stack_bottom);
145             maxsize = maxsize * 3/2;
146             stack.resize(maxsize);
147             stack_bottom = &stack[0];
148             stack_top = stack_bottom + sz;
149         }
150
151         int prev_flag = 0;
152         for (int j = 0; j < src.cols; j++)
153         {
154             #define CANNY_SHIFT 15
155             const int TG22 = (int)(0.4142135623730950488016887242097*(1<<
                    CANNY_SHIFT) + 0.5);
156
157             int m = _mag[j];
158
159             if (m > low)
160             {
161                 int xs = _x[j];
162                 int ys = _y[j];
163                 int x = std::abs(xs);
164                 int y = std::abs(ys) << CANNY_SHIFT;
165
166                 int tg22x = x * TG22;
167
168                 if (y < tg22x)
```

```
169                 {
170                     if (m > _mag[j-1] && m >= _mag[j+1]) goto
                            __ocv_canny_push;
171                 }
172                 else
173                 {
174                     int tg67x = tg22x + (x << (CANNY_SHIFT+1));
175                     if (y > tg67x)
176                     {
177                         if (m > _mag[j+magstep2] && m >= _mag[j+magstep1])
                                goto __ocv_canny_push;
178                     }
179                     else
180                     {
181                         int s = (xs ^ ys) < 0 ? -1 : 1;
182                         if (m > _mag[j+magstep2-s] && m > _mag[j+magstep1+s
                                ]) goto __ocv_canny_push;
183                     }
184                 }
185             }
186             prev_flag = 0;
187             _map[j] = uchar(1);
188             continue;
189 __ocv_canny_push:
190             if (!prev_flag && m > high && _map[j-mapstep] != 2)
191             {
192                 CANNY_PUSH(_map + j);
193                 prev_flag = 1;
194             }
195             else
196                 _map[j] = 0;
197         }
198
199         // scroll the ring buffer
200         _mag = mag_buf[0];
201         mag_buf[0] = mag_buf[1];
202         mag_buf[1] = mag_buf[2];
203         mag_buf[2] = _mag;
204     }
205
206 //     // now track the edges (hysteresis thresholding)
207 //     while (stack_top > stack_bottom)
208 //     {
209 //         uchar* m;
210 //         if ((stack_top - stack_bottom) + 8 > maxsize)
211 //         {
212 //             int sz = (int)(stack_top - stack_bottom);
```

```
213 //             maxsize = maxsize * 3/2;
214 //             stack.resize(maxsize);
215 //             stack_bottom = &stack[0];
216 //             stack_top = stack_bottom + sz;
217 //         }
218 //
219 //         CANNY_POP(m);
220 //
221 //         if (!m[-1])          CANNY_PUSH(m - 1);
222 //         if (!m[1])           CANNY_PUSH(m + 1);
223 //         if (!m[-mapstep-1]) CANNY_PUSH(m - mapstep - 1);
224 //         if (!m[-mapstep])    CANNY_PUSH(m - mapstep);
225 //         if (!m[-mapstep+1]) CANNY_PUSH(m - mapstep + 1);
226 //         if (!m[mapstep-1])  CANNY_PUSH(m + mapstep - 1);
227 //         if (!m[mapstep])     CANNY_PUSH(m + mapstep);
228 //         if (!m[mapstep+1])  CANNY_PUSH(m + mapstep + 1);
229 //     }
230
231     // the final pass, form the final image
232     const uchar* pmap = map + mapstep + 1;
233     uchar* pdst = dst.ptr();
234     for (int i = 0; i < src.rows; i++, pmap += mapstep, pdst += dst.step)
235     {
236         for (int j = 0; j < src.cols; j++){
237
238                 if(pmap[j] == 0)         // Candidato
239                         pdst[j] = 1;
240                 else if (pmap[j] == 1)   // Nao-borda
241                         pdst[j] = 0;
242                 else                     // Borda
243                         pdst[j] = 2;
244
245
246         }
247     }
248
249
250 }
```

B.2 Código-fonte do aplicativo de cálculo das taxas de avaliação qualitativas

```
1 #include "opencv2/imgproc/imgproc.hpp"
2 #include "opencv2/imgproc/imgproc_c.h"
3 #include "opencv2/highgui/highgui.hpp"
```

```
4  #include "opencv2/core/internal.hpp"
5  #include <math.h>
6  #include <assert.h>
7  #include <string.h>
8  #include <stdlib.h>
9  #include <stdio.h>
10 #include <limits.h>
11 #include <float.h>
12
13 using namespace cv;
14
15 // Comando para compilacao deste codigo
16 // g++ calcula.cpp -o calcula -std=c++11 `pkg-config --cflags --libs opencv
       `
17
18
19 int main( int argc, char** argv )
20 {
21
22         Mat srcA, srcB;
23
24         double co = 0; // Detectados corretamente
25         double nd = 0; // Nao Detectados corretamente
26         double fa = 0; // Falsos alarmes
27
28         double p_co = 0; // Detectados corretamente em porcentagem
29         double p_nd = 0; // Nao Detectados corretamente em porcentagem
30         double p_fa = 0; // Falsos alarmes em porcentagem
31
32         double sum_a = 0;
33         double sum_b = 0;
34         double max_ab = 0;
35
36         srcA = imread(argv[2], CV_LOAD_IMAGE_GRAYSCALE);
37         srcB = imread(argv[3], CV_LOAD_IMAGE_GRAYSCALE);
38
39         if( !srcA.data )
40                 return -1;
41
42         if( !srcB.data )
43                 return -1;
44
45
46         co = sum(srcA & srcB)[0];
47         nd = sum(srcA & (~srcB))[0];
48         fa = sum((~srcA) & srcB)[0];
49
```

```
50          max_ab = sum(max(srcA,srcB))[0];
51
52          p_co = co / max_ab;
53          p_nd = nd / max_ab;
54          p_fa = fa / max_ab;
55
56          printf("%s %.10f %.10f %.10f\n", argv[1], p_co, p_nd, p_fa);
57
58          return 0;
59
60  }
```

B.3 Função do algoritmo de Canny da biblioteca OpenCV com adições para medição do tempo gasto da histerese da própria biblioteca

```
1  int Canny( InputArray _src, OutputArray _dst,
2                 double low_thresh, double high_thresh,
3                 int aperture_size, bool L2gradient )
4  {
5
6      timespec start, finish;
7
8      Mat src = _src.getMat();
9      CV_Assert( src.depth() == CV_8U );
10
11     _dst.create(src.size(), CV_8U);
12     Mat dst = _dst.getMat();
13
14     if (!L2gradient && (aperture_size & CV_CANNY_L2_GRADIENT) ==
           CV_CANNY_L2_GRADIENT)
15     {
16         //backward compatibility
17         aperture_size &= ~CV_CANNY_L2_GRADIENT;
18         L2gradient = true;
19     }
20
21     if ((aperture_size & 1) == 0 || (aperture_size != -1 && (aperture_size
           < 3 || aperture_size > 7)))
22         CV_Error(CV_StsBadFlag, "");
23
24     if (low_thresh > high_thresh)
25         std::swap(low_thresh, high_thresh);
26
```

```
27 #ifdef HAVE_TEGRA_OPTIMIZATION
28     if (tegra::canny(src, dst, low_thresh, high_thresh, aperture_size,
           L2gradient))
29         return;
30 #endif
31
32 #ifdef USE_IPP_CANNY
33     if( aperture_size == 3 && !L2gradient &&
34         ippCanny(src, dst, (float)low_thresh, (float)high_thresh) )
35         return;
36 #endif
37
38     const int cn = src.channels();
39     Mat dx(src.rows, src.cols, CV_16SC(cn));
40     Mat dy(src.rows, src.cols, CV_16SC(cn));
41
42     Sobel(src, dx, CV_16S, 1, 0, aperture_size, 1, 0, cv::BORDER_REPLICATE)
           ;
43     Sobel(src, dy, CV_16S, 0, 1, aperture_size, 1, 0, cv::BORDER_REPLICATE)
           ;
44
45     if (L2gradient)
46     {
47         low_thresh = std::min(32767.0, low_thresh);
48         high_thresh = std::min(32767.0, high_thresh);
49
50         if (low_thresh > 0) low_thresh *= low_thresh;
51         if (high_thresh > 0) high_thresh *= high_thresh;
52     }
53     int low = cvFloor(low_thresh);
54     int high = cvFloor(high_thresh);
55
56     ptrdiff_t mapstep = src.cols + 2;
57     AutoBuffer<uchar> buffer((src.cols+2)*(src.rows+2) + cn * mapstep * 3 *
           sizeof(int));
58
59     int* mag_buf[3];
60     mag_buf[0] = (int*)(uchar*)buffer;
61     mag_buf[1] = mag_buf[0] + mapstep*cn;
62     mag_buf[2] = mag_buf[1] + mapstep*cn;
63     memset(mag_buf[0], 0, /* cn* */mapstep*sizeof(int));
64
65     uchar* map = (uchar*)(mag_buf[2] + mapstep*cn);
66     memset(map, 1, mapstep);
67     memset(map + mapstep*(src.rows + 1), 1, mapstep);
68
69     int maxsize = std::max(1 << 10, src.cols * src.rows / 10);
```

```
70      std::vector<uchar*> stack(maxsize);
71      uchar **stack_top = &stack[0];
72      uchar **stack_bottom = &stack[0];
73
74      /* sector numbers
75         (Top-Left Origin)
76
77          1   2   3
78           *  *  *
79            * * *
80          0*******0
81            * * *
82           *  *  *
83          3   2   1
84      */
85
86      #define CANNY_PUSH(d)     *(d) = uchar(2), *stack_top++ = (d)
87      #define CANNY_POP(d)      (d) = *--stack_top
88
89      // calculate magnitude and angle of gradient, perform non-maxima
            supression.
90      // fill the map with one of the following values:
91      //   0 - the pixel might belong to an edge
92      //   1 - the pixel can not belong to an edge
93      //   2 - the pixel does belong to an edge
94      for (int i = 0; i <= src.rows; i++)
95      {
96          int* _norm = mag_buf[(i > 0) + 1] + 1;
97          if (i < src.rows)
98          {
99              short* _dx = dx.ptr<short>(i);
100             short* _dy = dy.ptr<short>(i);
101
102             if (!L2gradient)
103             {
104                 for (int j = 0; j < src.cols*cn; j++)
105                     _norm[j] = std::abs(int(_dx[j])) + std::abs(int(_dy[j])
                            );
106             }
107             else
108             {
109                 for (int j = 0; j < src.cols*cn; j++)
110                     _norm[j] = int(_dx[j])*_dx[j] + int(_dy[j])*_dy[j];
111             }
112
113             if (cn > 1)
114             {
```

```
115                 for(int j = 0, jn = 0; j < src.cols; ++j, jn += cn)
116                 {
117                     int maxIdx = jn;
118                     for(int k = 1; k < cn; ++k)
119                         if(_norm[jn + k] > _norm[maxIdx]) maxIdx = jn + k;
120                     _norm[j] = _norm[maxIdx];
121                     _dx[j] = _dx[maxIdx];
122                     _dy[j] = _dy[maxIdx];
123                 }
124             }
125             _norm[-1] = _norm[src.cols] = 0;
126         }
127         else
128             memset(_norm-1, 0, /* cn* */mapstep*sizeof(int));
129
130         // at the very beginning we do not have a complete ring
131         // buffer of 3 magnitude rows for non-maxima suppression
132         if (i == 0)
133             continue;
134
135         uchar* _map = map + mapstep*i + 1;
136         _map[-1] = _map[src.cols] = 1;
137
138         int* _mag = mag_buf[1] + 1; // take the central row
139         ptrdiff_t magstep1 = mag_buf[2] - mag_buf[1];
140         ptrdiff_t magstep2 = mag_buf[0] - mag_buf[1];
141
142         const short* _x = dx.ptr<short>(i-1);
143         const short* _y = dy.ptr<short>(i-1);
144
145         if ((stack_top - stack_bottom) + src.cols > maxsize)
146         {
147             int sz = (int)(stack_top - stack_bottom);
148             maxsize = maxsize * 3/2;
149             stack.resize(maxsize);
150             stack_bottom = &stack[0];
151             stack_top = stack_bottom + sz;
152         }
153
154         int prev_flag = 0;
155         for (int j = 0; j < src.cols; j++)
156         {
157             #define CANNY_SHIFT 15
158             const int TG22 = (int)(0.4142135623730950488016887242097*(1<<
                    CANNY_SHIFT) + 0.5);
159
160             int m = _mag[j];
```

```
161
162             if (m > low)
163             {
164                 int xs = _x[j];
165                 int ys = _y[j];
166                 int x = std::abs(xs);
167                 int y = std::abs(ys) << CANNY_SHIFT;
168
169                 int tg22x = x * TG22;
170
171                 if (y < tg22x)
172                 {
173                     if (m > _mag[j-1] && m >= _mag[j+1]) goto
                            __ocv_canny_push;
174                 }
175                 else
176                 {
177                     int tg67x = tg22x + (x << (CANNY_SHIFT+1));
178                     if (y > tg67x)
179                     {
180                         if (m > _mag[j+magstep2] && m >= _mag[j+magstep1])
                                goto __ocv_canny_push;
181                     }
182                     else
183                     {
184                         int s = (xs ^ ys) < 0 ? -1 : 1;
185                         if (m > _mag[j+magstep2-s] && m > _mag[j+magstep1+s
                                ]) goto __ocv_canny_push;
186                     }
187                 }
188             }
189             prev_flag = 0;
190             _map[j] = uchar(1);
191             continue;
192 __ocv_canny_push:
193             if (!prev_flag && m > high && _map[j-mapstep] != 2)
194             {
195                 CANNY_PUSH(_map + j);
196                 prev_flag = 1;
197             }
198             else
199                 _map[j] = 0;
200         }
201
202         // scroll the ring buffer
203         _mag = mag_buf[0];
204         mag_buf[0] = mag_buf[1];
```

```
205         mag_buf[1] = mag_buf[2];
206         mag_buf[2] = _mag;
207     }
208
209 clock_gettime(CLOCK_MONOTONIC, &start);
210
211     // now track the edges (hysteresis thresholding)
212     while (stack_top > stack_bottom)
213     {
214         uchar* m;
215         if ((stack_top - stack_bottom) + 8 > maxsize)
216         {
217             int sz = (int)(stack_top - stack_bottom);
218             maxsize = maxsize * 3/2;
219             stack.resize(maxsize);
220             stack_bottom = &stack[0];
221             stack_top = stack_bottom + sz;
222         }
223
224         CANNY_POP(m);
225
226         if (!m[-1])         CANNY_PUSH(m - 1);
227         if (!m[1])          CANNY_PUSH(m + 1);
228         if (!m[-mapstep-1]) CANNY_PUSH(m - mapstep - 1);
229         if (!m[-mapstep])   CANNY_PUSH(m - mapstep);
230         if (!m[-mapstep+1]) CANNY_PUSH(m - mapstep + 1);
231         if (!m[mapstep-1])  CANNY_PUSH(m + mapstep - 1);
232         if (!m[mapstep])    CANNY_PUSH(m + mapstep);
233         if (!m[mapstep+1])  CANNY_PUSH(m + mapstep + 1);
234     }
235
236     // the final pass, form the final image
237     const uchar* pmap = map + mapstep + 1;
238     uchar* pdst = dst.ptr();
239     for (int i = 0; i < src.rows; i++, pmap += mapstep, pdst += dst.step)
240     {
241         for (int j = 0; j < src.cols; j++)
242             pdst[j] = (uchar)-(pmap[j] >> 1);
243     }
244
245 clock_gettime(CLOCK_MONOTONIC, &finish);
246
247 return diff(start,finish).tv_nsec;
248
249 }
250
251 timespec diff(timespec start, timespec end)
```

```
252 {
253         timespec temp;
254         if ((end.tv_nsec-start.tv_nsec)<0) {
255                 temp.tv_sec = end.tv_sec-start.tv_sec-1;
256                 temp.tv_nsec = 1000000000+end.tv_nsec-start.tv_nsec;
257         } else {
258                 temp.tv_sec = end.tv_sec-start.tv_sec;
259                 temp.tv_nsec = end.tv_nsec-start.tv_nsec;
260         }
261         return temp;
262 }
```

B.4 Código-fonte do *device driver* contido no *Memory Endpoint Test Driver* com alterações próprias deste trabalho

```
1  /*
2  //--
     ------------------------------------------------------------------------------

3  //--
4  //-- This file is owned and controlled by Xilinx and must be used solely
5  //-- for design, simulation, implementation and creation of design files
6  //-- limited to Xilinx devices or technologies. Use with non-Xilinx
7  //-- devices or technologies is expressly prohibited and immediately
8  //-- terminates your license.
9  //--
10 //-- Xilinx products are not intended for use in life support
11 //-- appliances, devices, or systems. Use in such applications is
12 //-- expressly prohibited.
13 //--
14 //--            **************************************
15 //--            ** Copyright (C) 2006, Xilinx, Inc. **
16 //--            ** All Rights Reserved.              **
17 //--            **************************************
18 //--
19 //--
     ------------------------------------------------------------------------------

20 //-- Filename: xpcie.c
21 //--
22 //-- Description: XPCIE device driver.
23 //--
24 //-- XPCIE is an example Red Hat device driver for the PCI Express Memory
```

```
25 //-- Endpoint Reference design. Device driver has been tested on fedora
26 //-- 2.6.18.
27 //--
28 //--
29 //--
30 //--
31 //--
32 //--
   ------------------------------------------------------------------------------

33 */
34
35 #include <linux/init.h>
36 #include <linux/module.h>
37 #include <linux/pci.h>
38 #include <linux/interrupt.h>
39 #include <linux/fs.h>
40 #include <asm/ioctl.h>
41 #include <asm/uaccess.h>   /* copy_to_user */
42
43 // semaphores
44 enum  {
45           SEM_READ,
46           SEM_WRITE,
47           SEM_WRITEREG,
48           SEM_READREG,
49           SEM_WAITFOR,
50           SEM_DMA,
51           NUM_SEMS
52 };
53
54 //semaphores
55 struct semaphore gSem[NUM_SEMS];
56
57
58 MODULE_LICENSE("Dual BSD/GPL");
59
60 // Max DMA Buffer Size
61
62 #define BUF_SIZE                    8194
63
64 #define PCI_VENDOR_ID_XILINX        0x10ee
65 //#define PCI_DEVICE_ID_XILINX_PCIE 0x0007
66 // PCI EXpress x8 Gen1
67 //#define PCI_DEVICE_ID_XILINX_PCIE 0x6018
68 // PCI EXpress x4 Gen2
69 #define PCI_DEVICE_ID_XILINX_PCIE 0x6024
```

```
70 //#define KINBURN_REGISTER_SIZE     (4*8)    // There are eight registers,
       and each is 4 bytes wide.
71 #define KINBURN_REGISTER_SIZE     (4 * 2048)    // There are eight
       registers, and each is 4 bytes wide.
72 #define HAVE_REGION               0x01      // I/O Memory region
73 #define HAVE_IRQ                  0x02      // Interupt
74 #define SUCCESS                   0
75 #define CRIT_ERR                  -1
76
77 //Status Flags:
78 //       1 = Resouce successfully acquired
79 //       0 = Resource not acquired.
80 #define HAVE_REGION 0x01               // I/O Memory region
81 #define HAVE_IRQ    0x02               // Interupt
82 #define HAVE_KREG   0x04               // Kernel registration
83
84 int             gDrvrMajor = 240;      // Major number not dynamic.
85 unsigned int    gStatFlags = 0x00;     // Status flags used for cleanup.
86 unsigned long   gBaseHdwr;             // Base register address (Hardware
       address)
87 unsigned long   gBaseLen;              // Base register address Length
88 void           *gBaseVirt = NULL;      // Base register address (Virtual
       address, for I/O).
89 char            gDrvrName[]= "xpcie";   // Name of driver in proc.
90 struct pci_dev *gDev = NULL;           // PCI device structure.
91 int             gIrq;                  // IRQ assigned by PCI system.
92 char           *gBufferUnaligned = NULL;   // Pointer to Unaligned DMA
       buffer.
93 char           *gReadBuffer      = NULL;   // Pointer to dword aligned DMA
       buffer.
94 char           *gWriteBuffer     = NULL;   // Pointer to dword aligned DMA
       buffer.
95
96
97 /*****************************************************************************

98  * Name:         XPCIe_Open
99  *
100 * Description: Book keeping routine invoked each time the device is opened
        .
101 *
102 * Arguments: inode :
103 *            filp  :
104 *
105 * Returns: 0 on success, error code on failure.
106 *
107 * Modification log:
```

```
108  * Date      Who  Description
109  * --------  ---  ------------------------------------------------------------
110  *
111  ****************************************************************************/

112  int XPCIe_Open(struct inode *inode, struct file *filp)
113  {
114      //MOD_INC_USE_COUNT;
115      printk("%s: Open: module opened\n",gDrvrName);
116      return SUCCESS;
117  }
118
119  /****************************************************************************

120   * Name:         XPCIe_Release
121   *
122   * Description: Book keeping routine invoked each time the device is closed
          .
123   *
124   * Arguments: inode :
125   *            filp  :
126   *
127   * Returns: 0 on success, error code on failure.
128   *
129   * Modification log:
130   * Date      Who  Description
131   * --------  ---  ------------------------------------------------------------
132   *
133   ****************************************************************************/

134  int XPCIe_Release(struct inode *inode, struct file *filp)
135  {
136      //MOD_DEC_USE_COUNT;
137      printk("%s: Release: module released\n",gDrvrName);
138      return(SUCCESS);
139  }
140
141  /****************************************************************************

142   * Name:         XPCIe_Write
143   *
144   * Description: This routine is invoked from user space to write data to
145   *              the 3GIO device.
146   *
147   * Arguments: filp  : file pointer to opened device.
```

```
148  *            buf    : pointer to location in users space, where data is to
149  *                     be acquired.
150  *            count  : Amount of data in bytes user wishes to send.
151  *
152  * Returns: SUCCESS  = Success
153  *          CRIT_ERR = Critical failure
154  *          TIME_ERR = Timeout
155  *          LINK_ERR = Link Failure
156  *
157  * Modification log:
158  * Date       Who  Description
159  * --------  ---
     ------------------------------------------------------------------------
160  *
161  ****************************************************************************/

162 ssize_t XPCIe_Write(struct file *filp, const char *buf, size_t count,
163                     loff_t *f_pos)
164 {
165         int ret = SUCCESS;
166         size_t size_wr = 4;
167
168         if(count < size_wr)
169                 memcpy((char *)gBaseVirt, buf, count);
170         else{
171                 int offset;
172
173                 for(offset=0; offset < count; offset += size_wr){
174                         memcpy((char *)(gBaseVirt + offset), (char *) (buf
                                + offset), size_wr);
175                 }
176
177                 if((count % size_wr) != 0){
178                         offset -= size_wr;
179                         memcpy((char *)(gBaseVirt + offset),(char *) (buf +
                                offset), count % size_wr);
180                 }
181         }
182
183
184         //memcpy((char *)gBaseVirt, buf, count);
185         printk("%s: XPCIe_Write: %d bytes have been written...\n",
                gDrvrName, (int) count);
186         return (ret);
187 }
188
```

```
189 /*****************************************************************************
190  * Name:         XPCIe_Read
191  *
192  * Description: This routine is invoked from user space to read data from
193  *              the 3GIO device. ***NOTE: This routine returns the entire
194  *              buffer, (BUF_SIZE), count is ignored!. The user App must
195  *              do any needed processing on the buffer.
196  *
197  * Arguments: filp  : file pointer to opened device.
198  *            buf   : pointer to location in users space, where data is to
199  *                    be placed.
200  *            count : Amount of data in bytes user wishes to read.
201  *
202  * Returns: SUCCESS  = Success
203  *          CRIT_ERR = Critical failure
204  *          TIME_ERR = Timeout
205  *          LINK_ERR = Link Failure
206  *
207  *
208  * Modification log:
209  * Date       Who  Description
210  * --------  ---  ------------------------------------------------------
211  *
212  *****************************************************************************/

213 ssize_t XPCIe_Read(struct file *filp, char *buf, size_t count, loff_t *
        f_pos)
214 {
215
216         size_t size_wr = 4;
217
218         if(count < size_wr)
219                 memcpy(buf, (char *)gBaseVirt, count);
220         else{
221                 int offset;
222
223                 for(offset=0; offset < count; offset += size_wr){
224                         memcpy((char *) (buf + offset), (char *)(gBaseVirt
                                + offset), size_wr);
225                 }
226
227                 if((count % size_wr) != 0){
228                         offset -= size_wr;
229                         memcpy((char *) (buf + offset), (char *)(gBaseVirt
                                + offset), count % size_wr);
```

```
230                 }
231         }
232
233         //memcpy(buf, (char *)gBaseVirt, count);
234         printk("%s: XPCIe_Read: %d bytes have been read...\n", gDrvrName, (
                int) count);
235         return (0);
236 }
237
238
239 /*****************************************************************************

240  * Name:         XPCIe_Ioctl
241  *
242  * Description: This routine is invoked from user space to configure the
243  *              running driver.
244  *
245  * Arguments: inode :
246  *            filp   : File pointer to opened device.
247  *            cmd    : Ioctl command to execute.
248  *            arg    : Argument to Ioctl command.
249  *
250  * Returns: 0 on success, error code on failure.
251  *
252  * Modification log:
253  * Date       Who  Description
254  * --------  ---
       ------------------------------------------------------------------
255  *
256  *****************************************************************************/

257 long XPCIe_Ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
258 {
259     long ret = SUCCESS;
260     return ret;
261 }
262
263
264 struct file_operations XPCIe_Intf = {
265     read:         XPCIe_Read,
266     write:        XPCIe_Write,
267     unlocked_ioctl:       XPCIe_Ioctl,
268     open:         XPCIe_Open,
269     release:      XPCIe_Release,
270 };
271
272
```

```
273 static int XPCIe_init(void)
274 {
275     //gDev = pci_find_device (PCI_VENDOR_ID_XILINX,
          PCI_DEVICE_ID_XILINX_PCIE, gDev);
276     gDev = pci_get_device (PCI_VENDOR_ID_XILINX, PCI_DEVICE_ID_XILINX_PCIE,
          gDev);
277     if (NULL == gDev) {
278         printk(/*KERN_WARNING*/"%s: Init: Hardware not found.\n", gDrvrName
              );
279         //return (CRIT_ERR);
280         return (-1);
281     }
282
283     if (0 > pci_enable_device(gDev)) {
284         printk(/*KERN_WARNING*/"%s: Init: Device not enabled.\n", gDrvrName
              );
285         //return (CRIT_ERR);
286         return (-1);
287     }
288
289     // Get Base Address of registers from pci structure. Should come from
          pci_dev
290     // structure, but that element seems to be missing on the development
          system.
291     gBaseHdwr = pci_resource_start (gDev, 0);
292     if (0 > gBaseHdwr) {
293         printk(/*KERN_WARNING*/"%s: Init: Base Address not set.\n",
              gDrvrName);
294         //return (CRIT_ERR);
295         return (-1);
296     }
297     printk(/*KERN_WARNING*/"%s: Base hw val %X\n", gDrvrName, (unsigned int
          )gBaseHdwr);
298
299     gBaseLen = pci_resource_len (gDev, 0);
300     printk(/*KERN_WARNING*/"%s: Base hw len %d\n", gDrvrName, (unsigned int
          )gBaseLen);
301
302     // Remap the I/O register block so that it can be safely accessed.
303     // I/O register block starts at gBaseHdwr and is 32 bytes long.
304     // It is cast to char because that is the way Linus does it.
305     // Reference "/usr/src/Linux-2.4/Documentation/IO-mapping.txt".
306
307     gBaseVirt = ioremap(gBaseHdwr, gBaseLen);
308     if (!gBaseVirt) {
309         printk(/*KERN_WARNING*/"%s: Init: Could not remap memory.\n",
              gDrvrName);
```

```
310         //return (CRIT_ERR);
311         return (-1);
312     }
313 
314     printk(/*KERN_WARNING*/"%s: Virt hw val %X\n", gDrvrName, (unsigned int
        )gBaseVirt);
315 
316     // Get IRQ from pci_dev structure. It may have been remapped by the
        kernel,
317     // and this value will be the correct one.
318 
319     gIrq = gDev->irq;
320     printk("%s: irq: %d\n", gDrvrName, gIrq);
321 
322     //--- START: Initialize Hardware
323 
324     // Try to gain exclusive control of memory for demo hardware.
325     if (0 > check_mem_region(gBaseHdwr, KINBURN_REGISTER_SIZE)) {
326         printk(/*KERN_WARNING*/"%s: Init: Memory in use.\n", gDrvrName);
327         //return (CRIT_ERR);
328         return (-1);
329     }
330 
331     request_mem_region(gBaseHdwr, KINBURN_REGISTER_SIZE, "3GIO_Demo_Drv");
332     gStatFlags = gStatFlags | HAVE_REGION;
333 
334     printk(/*KERN_WARNING*/"%s: Init:  Initialize Hardware Done..\n",
        gDrvrName);
335 
336 
337     //--- END: Initialize Hardware
338 
339     //--- START: Allocate Buffers
340 
341     gBufferUnaligned = kmalloc(BUF_SIZE, GFP_KERNEL);
342 
343     gReadBuffer = gBufferUnaligned;
344     if (NULL == gBufferUnaligned) {
345         printk(KERN_CRIT"%s: Init: Unable to allocate gBuffer.\n",gDrvrName
            );
346         return (-1);
347     }
348 
349     gWriteBuffer = kmalloc(BUF_SIZE, GFP_KERNEL);
350     if (NULL == gWriteBuffer) {
351         printk(KERN_CRIT"%s: Init: Unable to allocate gBuffer.\n",gDrvrName
            );
```

```
352             return (-1);
353         }
354 
355         //--- END: Allocate Buffers
356 
357         //--- START: Register Driver
358         // Register with the kernel as a character device.
359         // Abort if it fails.
360         if (0 > register_chrdev(gDrvrMajor, gDrvrName, &XPCIe_Intf)) {
361             printk(KERN_WARNING"%s: Init: will not register\n", gDrvrName);
362             return (CRIT_ERR);
363         }
364         printk(KERN_INFO"%s: Init: module registered\n", gDrvrName);
365         gStatFlags = gStatFlags | HAVE_KREG;
366 
367         printk("%s driver is loaded\n", gDrvrName);
368 
369       return 0;
370 
371 }
372 
373 static void XPCIe_exit(void)
374 {
375 
376   if (gStatFlags & HAVE_REGION) {
377      (void) release_mem_region(gBaseHdwr, KINBURN_REGISTER_SIZE);}
378 
379         // Release IRQ
380         if (gStatFlags & HAVE_IRQ) {
381             (void) free_irq(gIrq, gDev);
382         }
383 
384 
385         // Free buffer
386         if (NULL != gReadBuffer)
387             (void) kfree(gReadBuffer);
388         if (NULL != gWriteBuffer)
389             (void) kfree(gWriteBuffer);
390 
391         gReadBuffer = NULL;
392         gWriteBuffer = NULL;
393 
394 
395         if (gBaseVirt != NULL) {
396             iounmap(gBaseVirt);
397          }
398 
```

```
399     gBaseVirt = NULL;
400
401
402     // Unregister Device Driver
403     if (gStatFlags & HAVE_KREG)
404         unregister_chrdev(gDrvrMajor, gDrvrName);
405
406     gStatFlags = 0;
407
408   printk(/*KERN_ALERT*/ "%s driver is unloaded\n", gDrvrName);
409 }
410
411 module_init(XPCIe_init);
412 module_exit(XPCIe_exit);
```

ANEXO C

SCRIPTS PARA AUTOMAÇÃO DE PROCESSOS

C.1 Script conversor de imagens no formato JPEG para o formato PGM

```
#!/bin/bash

mogrify -format pgm \
        -colorspace gray \
        -compress none \
        -depth 8 \
        -thumbnail 64x64\! *.jpg

LIST_FILES=`ls -1 *pgm`

for tmp in $LIST_FILES; do

        cat $tmp | tr ' ' '\n' > temp_file
        cat temp_file | sed '/^$/ d' > $tmp
        rm temp_file

done
```

C.2 Script que aplica o algoritmo de cálculo das taxas qualitativas na base de imagens

```
#!/bin/bash

IMAGE_FILES=`ls -1 base_de_imagens/*.pgm`

HUMAN_OUTPUT_DIR=imagens_ideais
OPENCV_OUTPUT_DIR=canny_original
VHDL_OUTPUT_DIR=saida_simulada

echo -n > results_human_vhdl.txt
echo -n > results_human_opencv.txt
echo -n > results_opencv_vhdl.txt

for tmp in $IMAGE_FILES; do

```

```
15      image_file=`basename $tmp`
16      echo "./calcula $image_file $HUMAN_OUTPUT_DIR/$image_file
            $VHDL_OUTPUT_DIR/$image_file"
17      ./calcula $image_file $HUMAN_OUTPUT_DIR/$image_file $VHDL_OUTPUT_DIR/
            $image_file >> results_human_vhdl.txt
18
19      echo "./calcula $image_file $HUMAN_OUTPUT_DIR/$image_file
            $OPENCV_OUTPUT_DIR/$image_file"
20      ./calcula $image_file $HUMAN_OUTPUT_DIR/$image_file $OPENCV_OUTPUT_DIR/
            $image_file >> results_human_opencv.txt
21
22      echo "./calcula $image_file $OPENCV_OUTPUT_DIR/$image_file
            $VHDL_OUTPUT_DIR/$image_file"
23      ./calcula $image_file $OPENCV_OUTPUT_DIR/$image_file $VHDL_OUTPUT_DIR/
            $image_file >> results_opencv_vhdl.txt
24
25      echo "
            ------------------------------------------------------------------------------
            "
26  done
```

ANEXO D

TABELAS COM RESULTADOS POR IMAGEM

D.1 Tabela comparativa entre as bordas detectadas por humanos e pelo projeto proposto

Imagem	P_{co}	P_{nd}	P_{fa}
101085.pgm	0.1281302220	0.1753405225	0.6965292555
101087.pgm	0.1056480806	0.2611727077	0.6331792116
102061.pgm	0.0601264244	0.1367628712	0.8031107045
103070.pgm	0.0644111622	0.1542982412	0.7812905966
105025.pgm	0.0420005555	0.1251838746	0.8328155699
106024.pgm	0.0570826082	0.1570024955	0.7859148964
108005.pgm	0.0422580787	0.1384836452	0.8192582761
108070.pgm	0.0457686958	0.1888163760	0.7654149281
108082.pgm	0.0570318610	0.1849263831	0.7580417559
109053.pgm	0.0710075536	0.3388415467	0.5901681849
119082.pgm	0.0900349809	0.1692177333	0.7407472858
12084.pgm	0.0576488501	0.3440419905	0.5983091594
123074.pgm	0.0583722826	0.1009772735	0.8406504438
126007.pgm	0.1047844085	0.2479435749	0.6472720166
130026.pgm	0.0530263987	0.0961060528	0.8508675485
134035.pgm	0.0853151987	0.1654450262	0.7492397751
14037.pgm	0.1224932313	0.2647358736	0.6127708951
143090.pgm	0.0960463408	0.2206936207	0.6832600385
145086.pgm	0.0646387592	0.1806014143	0.7547598264
147091.pgm	0.1003516704	0.2153196433	0.6843286863
148026.pgm	0.0627665070	0.1035426280	0.8336908650
148089.pgm	0.0924293859	0.3257931845	0.5817883100
156065.pgm	0.0700314803	0.1350114696	0.7949570501
157055.pgm	0.0937005002	0.2088148362	0.6974901069
159008.pgm	0.0642113128	0.1168637103	0.8189337670
160068.pgm	0.0688948251	0.1744700531	0.7566351218
16077.pgm	0.0709580113	0.1471772160	0.7818647727
163085.pgm	0.0769695161	0.2567986732	0.6662318106

continua na próxima página

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
167062.pgm	0.1499931871	0.2355361766	0.6144706363
167083.pgm	0.0484048776	0.0748908837	0.8767042387
170057.pgm	0.0785585416	0.3312698274	0.5901716309
175032.pgm	0.0345155420	0.2821554573	0.6833290006
175043.pgm	0.0345022726	0.1037748774	0.8617228500
182053.pgm	0.0861287967	0.2565950115	0.6572761918
189080.pgm	0.0867370929	0.1526911382	0.7605843077
19021.pgm	0.0967911819	0.1405128421	0.7626959759
196073.pgm	0.0320136770	0.4784955248	0.4894907982
197017.pgm	0.1207272089	0.1802915862	0.6989812050
208001.pgm	0.0838364357	0.3175481690	0.5986153952
210088.pgm	0.0515132029	0.1356925055	0.8127942915
21077.pgm	0.0837492117	0.1504291316	0.7658216567
216081.pgm	0.1070670758	0.2636339947	0.6292989295
219090.pgm	0.0908104834	0.1468547723	0.7623347444
220075.pgm	0.0497334686	0.1205083866	0.8297581448
223061.pgm	0.0581363730	0.1318700443	0.8100153363
227092.pgm	0.0287808288	0.7242392468	0.2469799244
229036.pgm	0.0968702494	0.1550072024	0.7481225482
236037.pgm	0.0456699593	0.1555095149	0.7988205258
24077.pgm	0.0806007409	0.1744966983	0.7449025608
241004.pgm	0.1339219781	0.1839938383	0.6820841836
241048.pgm	0.0801074429	0.1771365390	0.7427620272
253027.pgm	0.0455072674	0.4685200623	0.4859726703
253055.pgm	0.0497677338	0.4397112367	0.5105210295
260058.pgm	0.0497143500	0.1639180480	0.7863676019
271035.pgm	0.0770375554	0.1786455084	0.7443169362
285079.pgm	0.0647673638	0.1686359326	0.7665967036
291000.pgm	0.0782540937	0.2291363865	0.6926095198
295087.pgm	0.0799252679	0.2591070271	0.6609677050
296007.pgm	0.0758439389	0.3202176797	0.6039383814
296059.pgm	0.1272585797	0.1478253747	0.7249160455
299086.pgm	0.0797201014	0.4112072751	0.5090726235
300091.pgm	0.1199680580	0.1563591769	0.7236727651
302008.pgm	0.1323730991	0.1965968380	0.6710300629
continua na próxima página			

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
304034.pgm	0.0327391155	0.0590265874	0.9082342970
304074.pgm	0.0701990351	0.1095387841	0.8202621808
306005.pgm	0.0717549001	0.1589260444	0.7693246270
3096.pgm	0.1514921711	0.1638711692	0.6846366597
33039.pgm	0.0449044596	0.0979620332	0.8571335072
351093.pgm	0.0860284390	0.1565755389	0.7573960221
361010.pgm	0.0932187707	0.2162789161	0.6905023133
37073.pgm	0.0964251679	0.2343954886	0.6691793435
376043.pgm	0.0597207220	0.1388256269	0.8014536510
38082.pgm	0.0507297397	0.1524935260	0.7967767343
38092.pgm	0.1109720587	0.2021745865	0.6868533548
385039.pgm	0.0958619443	0.2051296531	0.6990084026
41033.pgm	0.1048621982	0.1726009203	0.7225368816
41069.pgm	0.1037861715	0.1356531146	0.7605607140
42012.pgm	0.0653011862	0.1522880139	0.7824167575
42049.pgm	0.1363803626	0.1714144872	0.6922051503
43074.pgm	0.0848570754	0.4336876919	0.4814552327
45096.pgm	0.1084784537	0.2588706289	0.6326509174
54082.pgm	0.0795579417	0.2894493109	0.6309927475
55073.pgm	0.0794398419	0.2225142633	0.6980458948
58060.pgm	0.0611351790	0.1806703932	0.7581944279
62096.pgm	0.1052442851	0.1564589870	0.7382967279
65033.pgm	0.1047558044	0.2760564940	0.6191877017
66053.pgm	0.0652442618	0.2376647455	0.6970909927
69015.pgm	0.0903063487	0.1971924987	0.7125011526
69020.pgm	0.0821659973	0.1845883654	0.7332456374
69040.pgm	0.0209907641	0.8618130603	0.1171961756
76053.pgm	0.0626623415	0.3773027064	0.5600349521
78004.pgm	0.0936628866	0.2497489731	0.6565881403
8023.pgm	0.0023216868	0.3254357920	0.6722425211
85048.pgm	0.0972316971	0.1789555099	0.7238171382
86000.pgm	0.0761829532	0.1956850871	0.7281429308
86016.pgm	0.0855846733	0.6074905153	0.3069248113
86068.pgm	0.0978888169	0.2791649491	0.6229462340
87046.pgm	0.0528022233	0.1159181689	0.8312796078

continua na próxima página

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
89072.pgm	0.0709346045	0.1378263539	0.7912390416
97033.pgm	0.0688239325	0.1019940350	0.8291820325

D.2 Tabela comparativa entre as bordas detectadas por humanos e pela implementação inalterada do Canny pelo OpenCV

Imagem	P_{co}	P_{nd}	P_{fa}
101085.pgm	0.1281317624	0.1753426304	0.6965256071
101087.pgm	0.1056494945	0.2611762029	0.6331743026
102061.pgm	0.0601274246	0.1367651462	0.8031074292
103070.pgm	0.0643986709	0.1543107325	0.7812905966
105025.pgm	0.0420009875	0.1251851623	0.8328138501
106024.pgm	0.0570836595	0.1570053870	0.7859109535
108005.pgm	0.0422585422	0.1384851643	0.8192562935
108070.pgm	0.0457693487	0.1888190695	0.7654115818
108082.pgm	0.0570328306	0.1849295272	0.7580376422
109053.pgm	0.0710075536	0.3388415467	0.5901508997
119082.pgm	0.0900357968	0.1692192668	0.7407449363
12084.pgm	0.0576497740	0.3440475046	0.5983027214
123074.pgm	0.0583607988	0.1009887573	0.8406504438
126007.pgm	0.1047862701	0.2479479800	0.6472657499
130026.pgm	0.0530270060	0.0961071535	0.8508658405
134035.pgm	0.0853158580	0.1654463047	0.7492378373
14037.pgm	0.1224961028	0.2647420795	0.6127618177
143090.pgm	0.0960483862	0.2206983208	0.6832532930
145086.pgm	0.0646395770	0.1806036991	0.7547567239
147091.pgm	0.1003532459	0.2153230238	0.6843237303
148026.pgm	0.0627670566	0.1035435346	0.8336894088
148089.pgm	0.0924293859	0.3257931845	0.5817774296
156065.pgm	0.0700323348	0.1350131170	0.7949545482
157055.pgm	0.0937010103	0.2088159728	0.6974830169
159008.pgm	0.0642113128	0.1168637103	0.8189249769
160068.pgm	0.0688957762	0.1744724617	0.7566317621
16077.pgm	0.0709587272	0.1471787008	0.7818625720

continua na próxima página

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
163085.pgm	0.0769705844	0.2568022372	0.6662271784
167062.pgm	0.1500013627	0.2355490148	0.6144496225
167083.pgm	0.0484054221	0.0748917262	0.8767028517
170057.pgm	0.0785595954	0.3312742711	0.5901661335
175032.pgm	0.0345160548	0.2821596488	0.6833242964
175043.pgm	0.0345026836	0.1037761138	0.8617212026
182053.pgm	0.0861297794	0.2565979393	0.6572722812
189080.pgm	0.0867370929	0.1526911382	0.7605717689
19021.pgm	0.0967922014	0.1405143221	0.7626934764
196073.pgm	0.0320158235	0.4785276074	0.4894565691
197017.pgm	0.1207293295	0.1802947531	0.6989759174
208001.pgm	0.0838238712	0.3175607335	0.5986153952
210088.pgm	0.0515136303	0.1356936314	0.8127927382
21077.pgm	0.0837503599	0.1504311940	0.7658184461
216081.pgm	0.1070682146	0.2636367987	0.6292949867
219090.pgm	0.0908117626	0.1468568410	0.7623313964
220075.pgm	0.0497339425	0.1205095347	0.8297565228
223061.pgm	0.0581363730	0.1318700443	0.8099935827
227092.pgm	0.0287814575	0.7242550682	0.2469634743
229036.pgm	0.0968713525	0.1550089675	0.7481196800
236037.pgm	0.0456707044	0.1555120519	0.7988172438
24077.pgm	0.0806013899	0.1744981035	0.7449005065
241004.pgm	0.1339241049	0.1839967603	0.6820791348
241048.pgm	0.0801079243	0.1771376034	0.7427544723
253027.pgm	0.0455080921	0.4685285536	0.4859633543
253055.pgm	0.0497689835	0.4397222780	0.5105087384
260058.pgm	0.0497157492	0.1639226613	0.7863615896
271035.pgm	0.0770291236	0.1786539402	0.7443169362
285079.pgm	0.0647680539	0.1686377293	0.7665942168
291000.pgm	0.0782549586	0.2291389191	0.6926061222
295087.pgm	0.0799264868	0.2591109789	0.6609625342
296007.pgm	0.0758458023	0.3202255473	0.6039286504
296059.pgm	0.1272606644	0.1478277963	0.7249115392
299086.pgm	0.0797223599	0.4112189249	0.5090587152
300091.pgm	0.1199714795	0.1563636364	0.7236648841

continua na próxima página

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
302008.pgm	0.1323749991	0.1965996598	0.6710253411
304034.pgm	0.0327394273	0.0590271495	0.9082334232
304074.pgm	0.0701999217	0.1095401675	0.8202599108
306005.pgm	0.0717552999	0.1589269298	0.7693177703
3096.pgm	0.1514989293	0.1638784797	0.6846225910
33039.pgm	0.0448982263	0.0979682664	0.8571335072
351093.pgm	0.0860173213	0.1565866566	0.7573960221
361010.pgm	0.0932200029	0.2162817750	0.6904982221
37073.pgm	0.0964108370	0.2344098195	0.6691793435
376043.pgm	0.0597214992	0.1388274336	0.8014510672
38082.pgm	0.0507303451	0.1524953458	0.7967743090
38092.pgm	0.1109734338	0.2021770916	0.6868494746
385039.pgm	0.0958630270	0.2051319698	0.6990050032
41033.pgm	0.1048641879	0.1726041953	0.7225316167
41069.pgm	0.1037880427	0.1356555604	0.7605563969
42012.pgm	0.0653015752	0.1522889212	0.7824095036
42049.pgm	0.1363664767	0.1714283731	0.6922051503
43074.pgm	0.0848610849	0.4337081837	0.4814307314
45096.pgm	0.1084809861	0.2588766720	0.6326423419
54082.pgm	0.0795591906	0.2894538547	0.6309869547
55073.pgm	0.0794258239	0.2225282813	0.6980458948
58060.pgm	0.0611359478	0.1806726652	0.7581913870
62096.pgm	0.1052461721	0.1564617922	0.7382920357
65033.pgm	0.1047574612	0.2760608601	0.6191816787
66053.pgm	0.0652451995	0.2376681614	0.6970866391
69015.pgm	0.0903071418	0.1971942304	0.7124986279
69020.pgm	0.0821671471	0.1845909485	0.7332419043
69040.pgm	0.0209919012	0.8618597470	0.1171483518
76053.pgm	0.0626638417	0.3773117391	0.5600244191
78004.pgm	0.0936642020	0.2497524805	0.6565833175
8023.pgm	0.0023217752	0.3254481787	0.6722300461
85048.pgm	0.0972321196	0.1789562875	0.7238115929
86000.pgm	0.0761829532	0.1956850871	0.7281319598
86016.pgm	0.0855869844	0.6075069196	0.3069060960
86068.pgm	0.0978916993	0.2791731692	0.6229351315
continua na próxima página			

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
87046.pgm	0.0528027075	0.1159192321	0.8312780603
89072.pgm	0.0709352849	0.1378276758	0.7912370392
97033.pgm	0.0688246259	0.1019950627	0.8291803114

D.3 Tabela comparativa entre as bordas detectadas pelo implementação inalterada do Canny pelo OpenCV e pelo projeto proposto

Imagem	P_{co}	P_{nd}	P_{fa}
101085.pgm	0.9999854219	0.0000000000	0.0000145781
101087.pgm	0.9999818868	0.0000000000	0.0000181132
102061.pgm	0.9999807298	0.0000000000	0.0000192702
103070.pgm	0.9999852297	0.0000000000	0.0000147703
105025.pgm	0.9999882413	0.0000000000	0.0000117587
106024.pgm	0.9999781533	0.0000000000	0.0000218467
108005.pgm	0.9999872678	0.0000000000	0.0000127322
108070.pgm	0.9999824148	0.0000000000	0.0000175852
108082.pgm	0.9999791410	0.0000000000	0.0000208590
109053.pgm	0.9999738569	0.0000000000	0.0000261431
119082.pgm	0.9999890917	0.0000000000	0.0000109083
12084.pgm	0.9999755671	0.0000000000	0.0000244329
123074.pgm	0.9999872263	0.0000000000	0.0000127737
126007.pgm	0.9999763767	0.0000000000	0.0000236233
130026.pgm	0.9999873295	0.0000000000	0.0000126705
134035.pgm	0.9999907402	0.0000000000	0.0000092598
14037.pgm	0.9999681183	0.0000000000	0.0000318817
143090.pgm	0.9999726727	0.0000000000	0.0000273273
145086.pgm	0.9999845610	0.0000000000	0.0000154390
147091.pgm	0.9999799924	0.0000000000	0.0000200076
148026.pgm	0.9999902328	0.0000000000	0.0000097672
148089.pgm	0.9999838621	0.0000000000	0.0000161379
156065.pgm	0.9999858938	0.0000000000	0.0000141062
157055.pgm	0.9999862403	0.0000000000	0.0000137597
159008.pgm	0.9999900469	0.0000000000	0.0000099531
160068.pgm	0.9999832772	0.0000000000	0.0000167228
continua na próxima página			

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
16077.pgm	0.9999881704	0.0000000000	0.0000118296
163085.pgm	0.9999813262	0.0000000000	0.0000186738
167062.pgm	0.9999287038	0.0000000000	0.0000712962
167083.pgm	0.9999878402	0.0000000000	0.0000121598
170057.pgm	0.9999799412	0.0000000000	0.0000200588
175032.pgm	0.9999793061	0.0000000000	0.0000206939
175043.pgm	0.9999867067	0.0000000000	0.0000132933
182053.pgm	0.9999846516	0.0000000000	0.0000153484
189080.pgm	0.9999852018	0.0000000000	0.0000147982
19021.pgm	0.9999877452	0.0000000000	0.0000122548
196073.pgm	0.9998714405	0.0000000000	0.0001285595
197017.pgm	0.9999785711	0.0000000000	0.0000214289
208001.pgm	0.9999815892	0.0000000000	0.0000184108
210088.pgm	0.9999904002	0.0000000000	0.0000095998
21077.pgm	0.9999838621	0.0000000000	0.0000161379
216081.pgm	0.9999855561	0.0000000000	0.0000144439
219090.pgm	0.9999834884	0.0000000000	0.0000165116
220075.pgm	0.9999891671	0.0000000000	0.0000108329
223061.pgm	0.9999749427	0.0000000000	0.0000250573
227092.pgm	0.9999207827	0.0000000000	0.0000792173
229036.pgm	0.9999865240	0.0000000000	0.0000134760
236037.pgm	0.9999806823	0.0000000000	0.0000193177
24077.pgm	0.9999902449	0.0000000000	0.0000097551
241004.pgm	0.9999805385	0.0000000000	0.0000194615
241048.pgm	0.9999853947	0.0000000000	0.0000146053
253027.pgm	0.9999659006	0.0000000000	0.0000340994
253055.pgm	0.9999551841	0.0000000000	0.0000448159
260058.pgm	0.9999663396	0.0000000000	0.0000336604
271035.pgm	0.9999897342	0.0000000000	0.0000102658
285079.pgm	0.9999871846	0.0000000000	0.0000128154
291000.pgm	0.9999856617	0.0000000000	0.0000143383
295087.pgm	0.9999794148	0.0000000000	0.0000205852
296007.pgm	0.9999638578	0.0000000000	0.0000361422
296059.pgm	0.9999807770	0.0000000000	0.0000192230
299086.pgm	0.9999518849	0.0000000000	0.0000481151

continua na próxima página

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
300091.pgm	0.9999661945	0.0000000000	0.0000338055
302008.pgm	0.9999821344	0.0000000000	0.0000178656
304034.pgm	0.9999898799	0.0000000000	0.0000101201
304074.pgm	0.9999858173	0.0000000000	0.0000141827
306005.pgm	0.9999867516	0.0000000000	0.0000132484
3096.pgm	0.9999466482	0.0000000000	0.0000533518
33039.pgm	0.9999930898	0.0000000000	0.0000069102
351093.pgm	0.9999868184	0.0000000000	0.0000131816
361010.pgm	0.9999831333	0.0000000000	0.0000168667
37073.pgm	0.9999812816	0.0000000000	0.0000187184
376043.pgm	0.9999848882	0.0000000000	0.0000151118
38082.pgm	0.9999859192	0.0000000000	0.0000140808
38092.pgm	0.9999844693	0.0000000000	0.0000155307
385039.pgm	0.9999857916	0.0000000000	0.0000142084
41033.pgm	0.9999770674	0.0000000000	0.0000229326
41069.pgm	0.9999791410	0.0000000000	0.0000208590
42012.pgm	0.9999859444	0.0000000000	0.0000140556
42049.pgm	0.9999832414	0.0000000000	0.0000167586
43074.pgm	0.9999165693	0.0000000000	0.0000834307
45096.pgm	0.9999685024	0.0000000000	0.0000314976
54082.pgm	0.9999779071	0.0000000000	0.0000220929
55073.pgm	0.9999819701	0.0000000000	0.0000180299
58060.pgm	0.9999846516	0.0000000000	0.0000153484
62096.pgm	0.9999787453	0.0000000000	0.0000212547
65033.pgm	0.9999781533	0.0000000000	0.0000218467
66053.pgm	0.9999811467	0.0000000000	0.0000188533
69015.pgm	0.9999890613	0.0000000000	0.0000109387
69020.pgm	0.9999828381	0.0000000000	0.0000171619
69040.pgm	0.9996079969	0.0000000000	0.0003920031
76053.pgm	0.9999615547	0.0000000000	0.0000384453
78004.pgm	0.9999812816	0.0000000000	0.0000187184
8023.pgm	0.9999435777	0.0000000000	0.0000564223
85048.pgm	0.9999894156	0.0000000000	0.0000105844
86000.pgm	0.9999863599	0.0000000000	0.0000136401
86016.pgm	0.9999312053	0.0000000000	0.0000687947
continua na próxima página			

continuação da página anterior			
Imagem	P_{co}	P_{nd}	P_{fa}
86068.pgm	0.9999591520	0.0000000000	0.0000408480
87046.pgm	0.9999896256	0.0000000000	0.0000103744
89072.pgm	0.9999888751	0.0000000000	0.0000111249
97033.pgm	0.9999887796	0.0000000000	0.0000112204

D.4 Tabela com a quantidade de ciclos gastos por imagem

Imagem	Quantidade de ciclos de clock gastos
101085.pgm	249857
105025.pgm	286721
101087.pgm	417795
102061.pgm	462852
103070.pgm	548869
106024.pgm	270337
108082.pgm	282625
109053.pgm	348162
108070.pgm	380929
108005.pgm	991237
12084.pgm	397313
119082.pgm	495617
130026.pgm	512001
126007.pgm	528385
123074.pgm	913413
147091.pgm	286721
145086.pgm	446465
134035.pgm	495619
143090.pgm	626689
14037.pgm	684037
159008.pgm	266241
148026.pgm	290818
148089.pgm	561155
156065.pgm	749572
157055.pgm	1101829
167062.pgm	200705
16077.pgm	274433
continua na próxima página	

continuação da página anterior	
Imagem	Quantidade de ciclos de clock gastos
160068.pgm	311297
167083.pgm	348162
163085.pgm	647173
175043.pgm	524289
170057.pgm	557057
175032.pgm	561154
189080.pgm	593921
182053.pgm	753669
210088.pgm	163841
208001.pgm	323585
197017.pgm	442369
19021.pgm	475140
196073.pgm	479237
216081.pgm	294913
219090.pgm	585730
223061.pgm	614401
21077.pgm	618498
220075.pgm	733189
24077.pgm	282625
229036.pgm	315393
236037.pgm	319490
241004.pgm	512004
227092.pgm	720901
253055.pgm	421889
271035.pgm	430081
253027.pgm	458753
241048.pgm	516098
260058.pgm	589829
296059.pgm	327681
285079.pgm	434178
291000.pgm	561155
295087.pgm	667652
296007.pgm	958469
302008.pgm	196609
300091.pgm	356353
continua na próxima página	

continuação da página anterior	
Imagem	Quantidade de ciclos de clock gastos
299086.pgm	413697
304074.pgm	450561
304034.pgm	528389
3096.pgm	90113
306005.pgm	188417
33039.pgm	319491
361010.pgm	409601
351093.pgm	569349
385039.pgm	245761
37073.pgm	270338
38092.pgm	303105
376043.pgm	499716
38082.pgm	688133
42049.pgm	159745
41069.pgm	294913
43074.pgm	389122
41033.pgm	557059
42012.pgm	667653
62096.pgm	389121
54082.pgm	417793
45096.pgm	598018
58060.pgm	733185
55073.pgm	811013
69040.pgm	286721
65033.pgm	372738
69020.pgm	446465
69015.pgm	770049
66053.pgm	786437
85048.pgm	303105
8023.pgm	360449
76053.pgm	372737
86000.pgm	389123
78004.pgm	724997
86016.pgm	167937
86068.pgm	233474
continua na próxima página	

continuação da página anterior	
Imagem	Quantidade de ciclos de clock gastos
97033.pgm	327681
89072.pgm	364545
87046.pgm	442373

D.5 Tabela com o tempo gasto pela histerese da biblioteca OpenCV

Imagem	Tempo gasto (nanossegundos)
101085.pgm	21431
101087.pgm	17789
102061.pgm	18028
103070.pgm	21352
105025.pgm	22929
106024.pgm	14462
108005.pgm	23066
108070.pgm	18470
108082.pgm	14421
109053.pgm	14386
119082.pgm	23005
12084.pgm	14713
123074.pgm	22592
126007.pgm	12473
130026.pgm	22276
134035.pgm	25729
14037.pgm	10718
143090.pgm	11328
145086.pgm	17241
147091.pgm	17843
148026.pgm	23708
148089.pgm	18384
156065.pgm	23150
157055.pgm	21440
159008.pgm	24018
160068.pgm	21530
16077.pgm	22518
continua na próxima página	

continuação da página anterior	
Imagem	Tempo gasto (nanossegundos)
163085.pgm	18146
167062.pgm	26817
167083.pgm	22387
170057.pgm	18792
175032.pgm	15750
175043.pgm	22936
182053.pgm	22095
189080.pgm	21913
19021.pgm	22183
196073.pgm	8249
197017.pgm	14462
208001.pgm	17673
210088.pgm	25270
21077.pgm	17868
216081.pgm	21744
219090.pgm	21482
220075.pgm	23847
223061.pgm	14289
227092.pgm	8427
229036.pgm	21976
236037.pgm	18200
24077.pgm	25076
241004.pgm	18119
241048.pgm	21773
253027.pgm	13863
253055.pgm	10586
260058.pgm	11525
271035.pgm	24257
285079.pgm	33001
291000.pgm	21846
295087.pgm	20002
296007.pgm	12506
296059.pgm	14671
299086.pgm	10728
300091.pgm	12526
continua na próxima página	

continuação da página anterior	
Imagem	Tempo gasto (nanossegundos)
302008.pgm	18054
304034.pgm	27884
304074.pgm	21178
306005.pgm	21224
3096.pgm	11282
33039.pgm	31286
351093.pgm	21827
361010.pgm	17707
37073.pgm	18214
376043.pgm	21129
38082.pgm	22246
38092.pgm	18845
385039.pgm	21003
41033.pgm	14410
41069.pgm	20098
42012.pgm	21229
42049.pgm	17955
43074.pgm	8547
45096.pgm	10993
54082.pgm	17685
55073.pgm	19826
58060.pgm	21577
62096.pgm	14596
65033.pgm	14281
66053.pgm	16895
69015.pgm	24719
69020.pgm	18110
69040.pgm	7274
76053.pgm	10414
78004.pgm	17957
8023.pgm	9676
85048.pgm	23588
86000.pgm	21818
86016.pgm	8368
86068.pgm	10583
continua na próxima página	

continuação da página anterior	
Imagem	Tempo gasto (nanossegundos)
87046.pgm	43523
89072.pgm	22576
97033.pgm	22491

D.6 Tabela com a quantidade de ciclos líquidos da histerese na implementação proposta

Imagem	Quantidade de cíclos líquidos
101085.pgm	59
105025.pgm	68
101087.pgm	100
102061.pgm	111
103070.pgm	132
106024.pgm	64
108082.pgm	67
109053.pgm	83
108070.pgm	91
108005.pgm	240
12084.pgm	95
119082.pgm	119
130026.pgm	123
126007.pgm	127
123074.pgm	221
147091.pgm	68
145086.pgm	107
134035.pgm	119
143090.pgm	151
14037.pgm	165
159008.pgm	63
148026.pgm	69
148089.pgm	135
156065.pgm	181
157055.pgm	267
167062.pgm	47
16077.pgm	65
continua na próxima página	

continuação da página anterior	
Imagem	Quantidade de cíclos líquidos
160068.pgm	74
167083.pgm	83
163085.pgm	156
175043.pgm	126
170057.pgm	134
175032.pgm	135
189080.pgm	143
182053.pgm	182
210088.pgm	38
208001.pgm	77
197017.pgm	106
19021.pgm	114
196073.pgm	115
216081.pgm	70
219090.pgm	141
223061.pgm	148
21077.pgm	149
220075.pgm	177
24077.pgm	67
229036.pgm	75
236037.pgm	76
241004.pgm	123
227092.pgm	174
253055.pgm	101
271035.pgm	103
253027.pgm	110
241048.pgm	124
260058.pgm	142
296059.pgm	78
285079.pgm	104
291000.pgm	135
295087.pgm	161
296007.pgm	232
302008.pgm	46
300091.pgm	85
continua na próxima página	

continuação da página anterior	
Imagem	Quantidade de cíclos líquidos
299086.pgm	99
304074.pgm	108
304034.pgm	127
3096.pgm	20
306005.pgm	44
33039.pgm	76
361010.pgm	98
351093.pgm	137
385039.pgm	58
37073.pgm	64
38092.pgm	72
376043.pgm	120
38082.pgm	166
42049.pgm	37
41069.pgm	70
43074.pgm	93
41033.pgm	134
42012.pgm	161
62096.pgm	93
54082.pgm	100
45096.pgm	144
58060.pgm	177
55073.pgm	196
69040.pgm	68
65033.pgm	89
69020.pgm	107
69015.pgm	186
66053.pgm	190
85048.pgm	72
8023.pgm	86
76053.pgm	89
86000.pgm	93
78004.pgm	175
86016.pgm	39
86068.pgm	55

continua na próxima página

continuação da página anterior	
Imagem	Quantidade de cíclos líquidos
97033.pgm	78
89072.pgm	87
87046.pgm	106

D.7 Tabela com o tempo líquido estimado da histerese na implementação proposta

Imagem	Tempo estimado (milissegundos)
101085.pgm	0.00059
105025.pgm	0.00068
101087.pgm	0.00100
102061.pgm	0.00111
103070.pgm	0.00132
106024.pgm	0.00064
108082.pgm	0.00067
109053.pgm	0.00083
108070.pgm	0.00091
108005.pgm	0.00240
12084.pgm	0.00095
119082.pgm	0.00119
130026.pgm	0.00123
126007.pgm	0.00127
123074.pgm	0.00221
147091.pgm	0.00068
145086.pgm	0.00107
134035.pgm	0.00119
143090.pgm	0.00151
14037.pgm	0.00165
159008.pgm	0.00063
148026.pgm	0.00069
148089.pgm	0.00135
156065.pgm	0.00181
157055.pgm	0.00267
167062.pgm	0.00047
16077.pgm	0.00065
continua na próxima página	

continuação da página anterior	
Imagem	Tempo estimado (milissegundos)
160068.pgm	0.00074
167083.pgm	0.00083
163085.pgm	0.00156
175043.pgm	0.00126
170057.pgm	0.00134
175032.pgm	0.00135
189080.pgm	0.00143
182053.pgm	0.00182
210088.pgm	0.00038
208001.pgm	0.00077
197017.pgm	0.00106
19021.pgm	0.00114
196073.pgm	0.00115
216081.pgm	0.00070
219090.pgm	0.00141
223061.pgm	0.00148
21077.pgm	0.00149
220075.pgm	0.00177
24077.pgm	0.00067
229036.pgm	0.00075
236037.pgm	0.00076
241004.pgm	0.00123
227092.pgm	0.00174
253055.pgm	0.00101
271035.pgm	0.00103
253027.pgm	0.00110
241048.pgm	0.00124
260058.pgm	0.00142
296059.pgm	0.00078
285079.pgm	0.00104
291000.pgm	0.00135
295087.pgm	0.00161
296007.pgm	0.00232
302008.pgm	0.00046
300091.pgm	0.00085
continua na próxima página	

continuação da página anterior	
Imagem	Tempo estimado (milissegundos)
299086.pgm	0.00099
304074.pgm	0.00108
304034.pgm	0.00127
3096.pgm	0.00020
306005.pgm	0.00044
33039.pgm	0.00076
361010.pgm	0.00098
351093.pgm	0.00137
385039.pgm	0.00058
37073.pgm	0.00064
38092.pgm	0.00072
376043.pgm	0.00120
38082.pgm	0.00166
42049.pgm	0.00037
41069.pgm	0.00070
43074.pgm	0.00093
41033.pgm	0.00134
42012.pgm	0.00161
62096.pgm	0.00093
54082.pgm	0.00100
45096.pgm	0.00144
58060.pgm	0.00177
55073.pgm	0.00196
69040.pgm	0.00068
65033.pgm	0.00089
69020.pgm	0.00107
69015.pgm	0.00186
66053.pgm	0.00190
85048.pgm	0.00072
8023.pgm	0.00086
76053.pgm	0.00089
86000.pgm	0.00093
78004.pgm	0.00175
86016.pgm	0.00039
86068.pgm	0.00055
continua na próxima página	

continuação da página anterior	
Imagem	Tempo estimado (milissegundos)
97033.pgm	0.00078
89072.pgm	0.00087
87046.pgm	0.00106

ANEXO E

TRECHOS DE CÓDIGO-FONTE NA LINGUAGEM DE DESCRIÇÃO VHDL

E.1 Código-fonte bubble_eval.vhdl

```
1  use work.work_type.all;
2
3  ----------------------------------------
4  -------- Entity Declaration ------------
5  ----------------------------------------
6
7  entity bubble_eval is
8          port(
9                  image_in:in image_u2;
10                 reset:in bit;
11                 clk:in bit;
12                 image_out:out image_u2
13         );
14 end bubble_eval;
15
16 ------------------------------------------------
17 ---------------- Architecture ------------------
18 ------------------------------------------------
19
20 architecture behaviour_bubble_eval of bubble_eval is
21 begin
22
23         lines: for i in 0 to MAX_LIN - 1 generate
24         begin
25                 columns: for j in 0 to MAX_COL - 1 generate
26                 begin
27                         grid: entity work.bubble_eval_element(
                                   behaviour_bubble_eval_element)
28                                 port map(pixel_in => image_in(i,j),
29                                         reset => reset,
30                                         clk => clk,
31                                         pixel_out => image_out(i,j));
32
33                 end generate columns;
34         end generate lines;
35
```

```
36 end behaviour_bubble_eval;
```

E.2 Código-fonte bubble_eval_element.vhdl

```
1  use work.work_type.all;
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  ------------------------------------
8  -------- Entity Declaration --------
9  ------------------------------------
10
11 entity bubble_eval_element is
12         port(
13                 pixel_in:in uint2;
14                 reset:in bit;
15                 clk:in bit;
16                 pixel_out: out uint2
17         );
18 end bubble_eval_element;
19
20 ------------------------------------------
21 ------------- Architecture ---------------
22 ------------------------------------------
23
24 architecture behaviour_bubble_eval_element of bubble_eval_element is
25 signal buf: uint2;
26 begin
27
28         behav: process(reset,clk) is
29         begin
30
31                 if reset = '1' then
32                         buf <= (others => '0');
33                 elsif clk'event and clk = '1' then
34                         buf <= pixel_in;
35                 end if;
36
37
38
39         end process behav;
40
41         pixel_out <= buf;
```

```
42
43 end behaviour_bubble_eval_element;
```

E.3 Código-fonte hysteresis.vhdl

```
1  use work.work_type.all;
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  ----------------------------------------
8  -------- Entity Declaration ------------
9  ----------------------------------------
10
11 entity hysteresis is
12         port(
13                 pixel_in:in std_logic_vector(7 downto 0);
14                 tag_in:in std_logic_vector(7 downto 0);
15                 address_in_w:in std_logic_vector(15 downto 0);
16                 address_in_r:in std_logic_vector(15 downto 0);
17                 reset:in std_logic;
18                 clk:in std_logic;
19                 start:out std_logic;
20                 finish:out std_logic;
21                 tag_out:out std_logic_vector(7 downto 0);
22                 pixel_out:out std_logic_vector(7 downto 0)
23         );
24
25 end hysteresis;
26
27 ------------------------------------------------
28 ---------------- Architecture ------------------
29 ------------------------------------------------
30
31 architecture behaviour of hysteresis is
32         signal sig_pixel_in: uint8;
33         signal sig_tag_in: uint8;
34         signal sig_address_in_r: uint16;
35         signal sig_address_in_w: uint16;
36         signal sig_reset: bit;
37         signal sig_clk: bit;
38         --
39         signal sig_stp: image_u2;
40         signal sig_tag_stp: uint8;
```

```
41          signal sig_valid_stp: bit;
42          --
43          signal sig_hysteresis: image_u8;
44          signal sig_tag_hysteresis: uint8;
45          signal sig_start_hysteresis: bit;
46          --
47          signal sig_tag_out: uint8;
48          signal sig_pixel_out: uint8;
49          signal sig_start: bit;
50          signal sig_finish: bit;
51  begin
52          -- Input Signals ------------------------------------------------
53          sig_pixel_in    <= unsigned(pixel_in);
54          sig_tag_in      <= unsigned(tag_in);
55          sig_address_in_r<= unsigned(address_in_r);
56          sig_address_in_w<= unsigned(address_in_w);
57          sig_reset       <= to_bit(reset);
58          sig_clk         <= to_bit(clk);
59
60          -- Serial to Parallel Stage -------------------------------------
61          stp: entity work.serial_to_parallel(behaviour_serial_to_parallel)
62                  port map(
63                          sig_pixel_in,
64                          sig_address_in_w,
65                          sig_reset,
66                          sig_clk,
67                          sig_valid_stp,
68                          sig_stp
69                  );
70
71          tag_stp: entity work.bubble_tag(behaviour_bubble_tag)
72                  port map(
73                          sig_tag_in,
74                          NO_RETAIN,
75                          sig_reset,
76                          sig_clk,
77                          sig_tag_stp
78                  );
79
80          -- Hysteresis Stage ---------------------------------------------
81          hysteresis_pipeline: entity work.hysteresis_pipeline(
                behaviour_hysteresis_pipeline)
82                  port map(
83                          sig_stp,
84                          sig_tag_stp,
85                          sig_reset,
86                          sig_valid_stp,
```

```
87                          sig_start_hysteresis,
88                          sig_tag_hysteresis,
89                          sig_hysteresis
90                  );
91
92          -- Serial to Parallel Stage ------------------------------------------
93          pts: entity work.parallel_to_serial(behaviour_parallel_to_serial)
94                  port map(
95                          sig_hysteresis,
96                          sig_tag_hysteresis,
97                          sig_address_in_r,
98                          --sig_start_hysteresis,
99                          NO_RETAIN,
100                         sig_reset,
101                         sig_clk,
102                         sig_start,
103                         sig_finish,
104                         sig_tag_out,
105                         sig_pixel_out
106                 );
107
108         -- Output Signals ------------------------------------------------------
109         pixel_out       <= std_logic_vector(sig_pixel_out);
110         tag_out         <= std_logic_vector(sig_tag_out);
111         start           <= to_stdulogic(sig_start);
112         finish          <= to_stdulogic(sig_finish);
113
114 end behaviour;
```

E.4 Código-fonte hysteresis_check_pixel.vhdl

```
1  use work.work_type.all;
2
3  ----------------------------------
4  -------- Entity Declaration --------
5  ----------------------------------
6
7  entity hysteresis_check_pixel is
8          port(
9                  image_in:in image_u2;
10                 reset:in bit;
11                 clk:in bit;
12                 bit_array_out:out bit_array
13         );
14 end hysteresis_check_pixel;
```

```
15
16 ------------------------------------------------
17 ---------------- Architecture ------------------
18 ------------------------------------------------
19
20 architecture behaviour_hysteresis_check_pixel of hysteresis_check_pixel is
21 begin
22
23         lines: for i in 0 to MAX_LIN - 1 generate
24         begin
25                 columns: for j in 0 to MAX_COL - 1 generate
26                 begin
27                         north_west: if i = 0 and j = 0 generate
28                                 grid: entity work.
                                    hysteresis_check_pixel_element(
                                    behaviour_hysteresis_check_pixel_element
                                    )
29                                      port map(pixel_in => image_in(i,j),
30                                              pixel_n => NOT_CONTOUR,
31                                              pixel_s => image_in(i+1,j),
32                                              pixel_w => NOT_CONTOUR,
33                                              pixel_e => image_in(i,j+1),
34                                              pixel_nw => NOT_CONTOUR,
35                                              pixel_ne => NOT_CONTOUR,
36                                              pixel_se => image_in(i+1,j
                                                    +1),
37                                              pixel_sw => NOT_CONTOUR,
38                                              reset => reset,
39                                              clk => clk,
40                                              bit_out => bit_array_out(i,
                                                    j));
41                         end generate north_west;
42
43                         north: if i = 0 and j > 0 and j < MAX_COL - 1
                               generate
44                                 grid: entity work.
                                    hysteresis_check_pixel_element(
                                    behaviour_hysteresis_check_pixel_element
                                    )
45                                      port map(pixel_in => image_in(i,j),
46                                              pixel_n => NOT_CONTOUR,
47                                              pixel_s => image_in(i+1,j),
48                                              pixel_w => image_in(i,j-1),
49                                              pixel_e => image_in(i,j+1),
50                                              pixel_nw => NOT_CONTOUR,
51                                              pixel_ne => NOT_CONTOUR,
```

```
52                                      pixel_se => image_in(i+1,j
                                            +1),
53                                      pixel_sw => image_in(i+1,j
                                            -1),
54                                      reset => reset,
55                                      clk => clk,
56                                      bit_out => bit_array_out(i,
                                            j));
57                      end generate north;
58
59                      north_east: if i = 0 and j = MAX_COL - 1 generate
60                              grid: entity work.
                                    hysteresis_check_pixel_element(
                                    behaviour_hysteresis_check_pixel_element
                                    )
61                                          port map(pixel_in => image_in(i,j),
62                                                  pixel_n => NOT_CONTOUR,
63                                                  pixel_s => image_in(i+1,j),
64                                                  pixel_w => image_in(i,j-1),
65                                                  pixel_e => NOT_CONTOUR,
66                                                  pixel_nw => NOT_CONTOUR,
67                                                  pixel_ne => NOT_CONTOUR,
68                                                  pixel_se => NOT_CONTOUR,
69                                                  pixel_sw => image_in(i+1,j
                                                        -1),
70                                                  reset => reset,
71                                                  clk => clk,
72                                                  bit_out => bit_array_out(i,
                                                        j));
73                      end generate north_east;
74
75                      south_west: if i = MAX_LIN - 1 and j = 0 generate
76                              grid: entity work.
                                    hysteresis_check_pixel_element(
                                    behaviour_hysteresis_check_pixel_element
                                    )
77                                          port map(pixel_in => image_in(i,j),
78                                                  pixel_n => image_in(i-1,j),
79                                                  pixel_s => NOT_CONTOUR,
80                                                  pixel_w => NOT_CONTOUR,
81                                                  pixel_e => image_in(i,j+1),
82                                                  pixel_nw => NOT_CONTOUR,
83                                                  pixel_ne => image_in(i-1,j
                                                        +1),
84                                                  pixel_se => NOT_CONTOUR,
85                                                  pixel_sw => NOT_CONTOUR,
86                                                  reset => reset,
```

```
87                                      clk => clk,
88                                      bit_out => bit_array_out(i,
                                            j));
89                          end generate south_west;
90
91                          south: if i = MAX_LIN - 1 and j > 0 and j < MAX_COL
                                - 1 generate
92                              grid: entity work.
                                    hysteresis_check_pixel_element(
                                    behaviour_hysteresis_check_pixel_element
                                    )
93                                      port map(pixel_in => image_in(i,j),
94                                              pixel_n => image_in(i-1,j),
95                                              pixel_s => NOT_CONTOUR,
96                                              pixel_w => image_in(i,j-1),
97                                              pixel_e => image_in(i,j+1),
98                                              pixel_nw => image_in(i-1,j
                                                    -1),
99                                              pixel_ne => image_in(i-1,j
                                                    +1),
100                                             pixel_se => NOT_CONTOUR,
101                                             pixel_sw => NOT_CONTOUR,
102                                             reset => reset,
103                                             clk => clk,
104                                             bit_out => bit_array_out(i,
                                                    j));
105                         end generate south;
106
107                         south_east: if i = MAX_LIN - 1 and j = MAX_COL - 1
                                generate
108                             grid: entity work.
                                    hysteresis_check_pixel_element(
                                    behaviour_hysteresis_check_pixel_element
                                    )
109                                     port map(pixel_in => image_in(i,j),
110                                             pixel_n => image_in(i-1,j),
111                                             pixel_s => NOT_CONTOUR,
112                                             pixel_w => image_in(i,j-1),
113                                             pixel_e => NOT_CONTOUR,
114                                             pixel_nw => image_in(i-1,j
                                                    -1),
115                                             pixel_ne => NOT_CONTOUR,
116                                             pixel_se => NOT_CONTOUR,
117                                             pixel_sw => NOT_CONTOUR,
118                                             reset => reset,
119                                             clk => clk,
```

```
120                                     bit_out => bit_array_out(i,
                                            j));
121                         end generate south_east;
122
123                         west: if i > 0 and i < MAX_LIN - 1 and j = 0
                                generate
124                                 grid: entity work.
                                        hysteresis_check_pixel_element(
                                        behaviour_hysteresis_check_pixel_element
                                        )
125                                         port map(pixel_in => image_in(i,j),
126                                                 pixel_n => image_in(i-1,j),
127                                                 pixel_s => image_in(i+1,j),
128                                                 pixel_w => NOT_CONTOUR,
129                                                 pixel_e => image_in(i,j+1),
130                                                 pixel_nw => NOT_CONTOUR,
131                                                 pixel_ne => image_in(i-1,j
                                                        +1),
132                                                 pixel_se => image_in(i+1,j
                                                        +1),
133                                                 pixel_sw => NOT_CONTOUR,
134                                                 reset => reset,
135                                                 clk => clk,
136                                                 bit_out => bit_array_out(i,
                                                        j));
137                         end generate west;
138
139                         east: if i > 0 and i < MAX_LIN - 1 and j = MAX_COL
                                - 1 generate
140                                 grid: entity work.
                                        hysteresis_check_pixel_element(
                                        behaviour_hysteresis_check_pixel_element
                                        )
141                                         port map(pixel_in => image_in(i,j),
142                                                 pixel_n => image_in(i-1,j),
143                                                 pixel_s => image_in(i+1,j),
144                                                 pixel_w => image_in(i,j-1),
145                                                 pixel_e => NOT_CONTOUR,
146                                                 pixel_nw => image_in(i-1,j
                                                        -1),
147                                                 pixel_ne => NOT_CONTOUR,
148                                                 pixel_se => NOT_CONTOUR,
149                                                 pixel_sw => image_in(i+1,j
                                                        -1),
150                                                 reset => reset,
151                                                 clk => clk,
```

```
152                                                 bit_out => bit_array_out(i,
                                                        j));
153                         end generate east;
154
155                         center: if i > 0 and i < MAX_LIN - 1 and j > 0 and
                                j < MAX_COL - 1 generate
156                                 grid: entity work.
                                        hysteresis_check_pixel_element(
                                        behaviour_hysteresis_check_pixel_element
                                        )
157                                         port map(pixel_in => image_in(i,j),
158                                                 pixel_n => image_in(i-1,j),
159                                                 pixel_s => image_in(i+1,j),
160                                                 pixel_w => image_in(i,j-1),
161                                                 pixel_e => image_in(i,j+1),
162                                                 pixel_nw => image_in(i-1,j
                                                        -1),
163                                                 pixel_ne => image_in(i-1,j
                                                        +1),
164                                                 pixel_se => image_in(i+1,j
                                                        +1),
165                                                 pixel_sw => image_in(i+1,j
                                                        -1),
166                                                 reset => reset,
167                                                 clk => clk,
168                                                 bit_out => bit_array_out(i,
                                                        j));
169                         end generate center;
170
171                 end generate columns;
172         end generate lines;
173
174 end behaviour_hysteresis_check_pixel;
```

E.5 Código-fonte hysteresis_check_pixel_element.vhdl

```
1  use work.work_type.all;
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  ----------------------------------------
8  -------- Entity Declaration ------------
9  ----------------------------------------
```

```
10
11 entity hysteresis_check_pixel_element is
12         port(
13                 pixel_in:in uint2;
14                 pixel_n:in uint2;
15                 pixel_s:in uint2;
16                 pixel_w:in uint2;
17                 pixel_e:in uint2;
18                 pixel_nw:in uint2;
19                 pixel_ne:in uint2;
20                 pixel_se:in uint2;
21                 pixel_sw:in uint2;
22                 reset:in bit;
23                 clk:in bit;
24                 bit_out: out bit
25         );
26 end hysteresis_check_pixel_element;
27
28 ----------------------------------------
29 ------------ Architecture --------------
30 ----------------------------------------
31
32 architecture behaviour_hysteresis_check_pixel_element of
       hysteresis_check_pixel_element is
33 signal buf: bit;
34 begin
35
36         behav: process(reset,clk) is
37         variable sum: uint2;
38         begin
39
40                 if reset = '1' then
41                         buf <= '0';
42                 elsif clk'event and clk = '1' then
43                         -- Truque aqui
44                         -- Eh feito o somatorio de todos os vizinhos.
45                         sum :=  ((pixel_n or pixel_s) or
46                                  (pixel_w or pixel_e)) or
47                                 ((pixel_nw or pixel_ne) or
48                                 (pixel_se or pixel_sw));
49
50                         if pixel_in = CANDIDATE then
51                                 buf <= to_bit(sum(1));
52                         else
53                                 buf <= '0';
54                         end if;
55                 end if;
```

```
56
57          end process behav;
58
59          bit_out <= buf;
60
61
62  end behaviour_hysteresis_check_pixel_element;
```

E.6 Código-fonte hysteresis_out.vhdl

```
1   use work.work_type.all;
2
3   ---------------------------------------
4   -------- Entity Declaration ----------
5   ---------------------------------------
6
7   entity hysteresis_out is
8           port(
9                   image_eval:in image_u2;
10                  reset:in bit;
11                  clk:in bit;
12                  image_out:out image_u8
13          );
14  end hysteresis_out;
15
16  -----------------------------------------------
17  ------------- Architecture --------------------
18  -----------------------------------------------
19
20  architecture behaviour_hysteresis_out of hysteresis_out is
21  begin
22
23          lines: for i in 0 to MAX_LIN - 1 generate
24          begin
25                  columns: for j in 0 to MAX_COL - 1 generate
26                  begin
27                          grid: entity work.hysteresis_out_element(
                                behav_hysteresis_out_element)
28                                  port map(
29                                          pixel_eval => image_eval(i,j),
30                                          reset => reset,
31                                          clk => clk,
32                                          pixel_out => image_out(i,j)
33                                  );
34
```

```
35                  end generate columns;
36          end generate lines;
37
38  end behaviour_hysteresis_out;
```

E.7 Código-fonte hysteresis_out_element.vhdl

```
1   use work.work_type.all;
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.numeric_std.all;
6
7   ------------------------------------
8   -------- Entity Declaration --------
9   ------------------------------------
10
11  entity hysteresis_out_element is
12          port(
13                  pixel_eval:in uint2;
14                  reset:in bit;
15                  clk:in bit;
16                  pixel_out: out uint8
17          );
18  end hysteresis_out_element;
19
20  ------------------------------------------
21  ------------ Architecture ----------------
22  ------------------------------------------
23
24  architecture behav_hysteresis_out_element of hysteresis_out_element is
25  signal buf: uint8;
26  begin
27
28          behav: process(reset,clk) is
29          begin
30                  if reset = '1' then
31                          buf <= (others => '0');
32                  elsif clk'event and clk = '1' then
33
34                          if pixel_eval = CONTOUR then
35                                  buf <= RANGE_UINT8;
36                          else
37                                  buf <= (others => '0');
38                          end if;
```

```
39
40                  end if;
41
42          end process behav;
43
44          pixel_out <= buf;
45
46
47  end behav_hysteresis_out_element;
```

E.8 Código-fonte hysteresis_pipeline.vhdl

```
1   use work.work_type.all;
2
3   ----------------------------------
4   ------- Entity Declaration -------
5   ----------------------------------
6
7   entity hysteresis_pipeline is
8           port(
9                   image_in:in image_u2;
10                  tag_in:in uint8;
11                  reset:in bit;
12                  clk:in bit;
13                  ready:out bit;
14                  tag_out:out uint8;
15                  image_out:out image_u8
16          );
17  end hysteresis_pipeline;
18
19  ------------------------------------------
20  ------------ Architecture ----------------
21  ------------------------------------------
22
23  architecture behaviour_hysteresis_pipeline of hysteresis_pipeline is
24          signal sig_mux_image_eval: image_u2;
25          signal sig_tag_mux_image_eval: uint8;
26          --
27          signal sig_bubble_mux_image_eval_first: image_u2;
28          signal sig_hysteresis_check_pixel: bit_array;
29          signal sig_tag_hysteresis_check_pixel: uint8;
30          --
31          signal sig_bubble_mux_image_eval_second: image_u2;
32          signal sig_hysteresis_sum_first: bit_array_div_4;
33          signal sig_tag_hysteresis_sum_first: uint8;
```

```
34          --
35          signal sig_bubble_mux_image_eval_third: image_u2;
36          signal sig_hysteresis_sum_second: bit_array_div_16;
37          signal sig_tag_hysteresis_sum_second: uint8;
38          --
39          signal sig_hysteresis_out: image_u8;
40          signal sig_hysteresis_sum_third: bit;
41          signal sig_hysteresis_spread_elect: image_u2;
42          signal sig_tag_hysteresis_spread_elect: uint8;
43  begin
44
45          -- Mux hysteresis pipe
46          mux_image_eval: entity work.mux(behaviour_mux)
47                  port map(
48                          image_in,
49                          sig_hysteresis_spread_elect,
50                          sig_hysteresis_sum_third,
51                          reset,
52                          clk,
53                          sig_mux_image_eval
54                  );
55
56          tag_mux_image_eval: entity work.mux_tag(behaviour_mux_tag)
57                  port map(
58                          tag_in,
59                          sig_tag_hysteresis_spread_elect,
60                          sig_hysteresis_sum_third,
61                          reset,
62                          clk,
63                          sig_tag_mux_image_eval
64                  );
65
66          -- Check pixel hysteresis pipe
67          bubble_mux_image_eval_first: entity work.bubble_eval(
                behaviour_bubble_eval)
68                  port map(
69                          sig_mux_image_eval,
70                          reset,
71                          clk,
72                          sig_bubble_mux_image_eval_first
73                  );
74
75          hysteresis_check_pixel: entity work.hysteresis_check_pixel(
                behaviour_hysteresis_check_pixel)
76                  port map(
77                          sig_mux_image_eval,
78                          reset,
```

```
79                      clk,
80                      sig_hysteresis_check_pixel
81              );
82
83          tag_hysteresis_check_pixel: entity work.bubble_tag(
                behaviour_bubble_tag)
84              port map(
85                      sig_tag_mux_image_eval,
86                      NO_RETAIN,
87                      reset,
88                      clk,
89                      sig_tag_hysteresis_check_pixel
90              );
91
92          -- First sum hysteresis pipe
93          bubble_mux_image_eval_second: entity work.bubble_eval(
                behaviour_bubble_eval)
94              port map(
95                      sig_bubble_mux_image_eval_first,
96                      reset,
97                      clk,
98                      sig_bubble_mux_image_eval_second
99              );
100
101         hysteresis_sum_first: entity work.hysteresis_sum_first(
                behaviour_hysteresis_sum_first)
102             port map(
103                     sig_hysteresis_check_pixel,
104                     reset,
105                     clk,
106                     sig_hysteresis_sum_first
107             );
108
109         tag_hysteresis_sum_first: entity work.bubble_tag(
                behaviour_bubble_tag)
110             port map(
111                     sig_tag_hysteresis_check_pixel,
112                     NO_RETAIN,
113                     reset,
114                     clk,
115                     sig_tag_hysteresis_sum_first
116             );
117
118         -- Second sum hysteresis pipe
119         bubble_mux_image_eval_third: entity work.bubble_eval(
                behaviour_bubble_eval)
120             port map(
```

```
121                     sig_bubble_mux_image_eval_second,
122                     reset,
123                     clk,
124                     sig_bubble_mux_image_eval_third
125             );
126
127         hysteresis_sum_second: entity work.hysteresis_sum_second(
                behaviour_hysteresis_sum_second)
128             port map(
129                     sig_hysteresis_sum_first,
130                     reset,
131                     clk,
132                     sig_hysteresis_sum_second
133             );
134
135         tag_hysteresis_sum_second: entity work.bubble_tag(
                behaviour_bubble_tag)
136             port map(
137                     sig_tag_hysteresis_sum_first,
138                     NO_RETAIN,
139                     reset,
140                     clk,
141                     sig_tag_hysteresis_sum_second
142             );
143
144         -- Third sum and spread elect hysteresis pipe
145         hysteresis_out: entity work.hysteresis_out(behaviour_hysteresis_out
                )
146             port map(
147                     sig_bubble_mux_image_eval_third,
148                     reset,
149                     clk,
150                     sig_hysteresis_out
151             );
152
153         hysteresis_sum_third: entity work.hysteresis_sum_third(
                behaviour_hysteresis_sum_third)
154             port map(
155                     sig_hysteresis_sum_second,
156                     reset,
157                     clk,
158                     sig_hysteresis_sum_third
159             );
160
161         hysteresis_spread_elect: entity work.hysteresis_spread_elect(
                behaviour_hysteresis_spread_elect)
162             port map(
```

```
163                         sig_bubble_mux_image_eval_third,
164                         reset,
165                         clk,
166                         sig_hysteresis_spread_elect
167                 );
168
169         tag_hysteresis_spread_elect: entity work.bubble_tag(
                behaviour_bubble_tag)
170                 port map(
171                         sig_tag_hysteresis_sum_second,
172                         NO_RETAIN,
173                         reset,
174                         clk,
175                         sig_tag_hysteresis_spread_elect
176                 );
177
178         -- Output Signals
179         image_out       <= sig_hysteresis_out;
180         tag_out         <= sig_tag_hysteresis_spread_elect;
181         ready           <= sig_hysteresis_sum_third;
182
183 end behaviour_hysteresis_pipeline;
```

E.9 Código-fonte hysteresis_spread_elect.vhdl

```
1  use work.work_type.all;
2
3  ----------------------------------
4  -------- Entity Declaration --------
5  ----------------------------------
6
7  entity hysteresis_spread_elect is
8          port(
9                  image_in:in image_u2;
10                 reset:in bit;
11                 clk:in bit;
12                 image_out:out image_u2
13         );
14 end hysteresis_spread_elect;
15
16 ------------------------------------------
17 ------------- Architecture -------------
18 ------------------------------------------
19
```

```
20 architecture behaviour_hysteresis_spread_elect of hysteresis_spread_elect
       is
21 begin
22
23         lines: for i in 0 to MAX_LIN - 1 generate
24         begin
25                 columns: for j in 0 to MAX_COL - 1 generate
26                 begin
27                         north_west: if i = 0 and j = 0 generate
28                                 grid: entity work.
                                       hysteresis_spread_elect_element(
                                       behaviour_hysteresis_spread_elect_element
                                       )
29                                       port map(pixel_in => image_in(i,j),
30                                                pixel_n => NOT_CONTOUR,
31                                                pixel_s => image_in(i+1,j),
32                                                pixel_w => NOT_CONTOUR,
33                                                pixel_e => image_in(i,j+1),
34                                                pixel_nw => NOT_CONTOUR,
35                                                pixel_ne => NOT_CONTOUR,
36                                                pixel_se => image_in(i+1,j
                                                     +1),
37                                                pixel_sw => NOT_CONTOUR,
38                                                reset => reset,
39                                                clk => clk,
40                                                pixel_out => image_out(i,j)
                                                     );
41                         end generate north_west;
42
43                         north: if i = 0 and j > 0 and j < MAX_COL - 1
                               generate
44                                 grid: entity work.
                                       hysteresis_spread_elect_element(
                                       behaviour_hysteresis_spread_elect_element
                                       )
45                                       port map(pixel_in => image_in(i,j),
46                                                pixel_n => NOT_CONTOUR,
47                                                pixel_s => image_in(i+1,j),
48                                                pixel_w => image_in(i,j-1),
49                                                pixel_e => image_in(i,j+1),
50                                                pixel_nw => NOT_CONTOUR,
51                                                pixel_ne => NOT_CONTOUR,
52                                                pixel_se => image_in(i+1,j
                                                     +1),
53                                                pixel_sw => image_in(i+1,j
                                                     -1),
54                                                reset => reset,
```

```
55                                      clk => clk,
56                                      pixel_out => image_out(i,j)
                                            );
57                          end generate north;
58
59                          north_east: if i = 0 and j = MAX_COL - 1 generate
60                                  grid: entity work.
                                        hysteresis_spread_elect_element(
                                        behaviour_hysteresis_spread_elect_element
                                        )
61                                          port map(pixel_in => image_in(i,j),
62                                                  pixel_n => NOT_CONTOUR,
63                                                  pixel_s => image_in(i+1,j),
64                                                  pixel_w => image_in(i,j-1),
65                                                  pixel_e => NOT_CONTOUR,
66                                                  pixel_nw => NOT_CONTOUR,
67                                                  pixel_ne => NOT_CONTOUR,
68                                                  pixel_se => NOT_CONTOUR,
69                                                  pixel_sw => image_in(i+1,j
                                                        -1),
70                                                  reset => reset,
71                                                  clk => clk,
72                                                  pixel_out => image_out(i,j)
                                                        );
73                          end generate north_east;
74
75                          south_west: if i = MAX_LIN - 1 and j = 0 generate
76                                  grid: entity work.
                                        hysteresis_spread_elect_element(
                                        behaviour_hysteresis_spread_elect_element
                                        )
77                                          port map(pixel_in => image_in(i,j),
78                                                  pixel_n => image_in(i-1,j),
79                                                  pixel_s => NOT_CONTOUR,
80                                                  pixel_w => NOT_CONTOUR,
81                                                  pixel_e => image_in(i,j+1),
82                                                  pixel_nw => NOT_CONTOUR,
83                                                  pixel_ne => image_in(i-1,j
                                                        +1),
84                                                  pixel_se => NOT_CONTOUR,
85                                                  pixel_sw => NOT_CONTOUR,
86                                                  reset => reset,
87                                                  clk => clk,
88                                                  pixel_out => image_out(i,j)
                                                        );
89                          end generate south_west;
90
```

```
91                  south: if i = MAX_LIN - 1 and j > 0 and j < MAX_COL
                        - 1 generate
92                      grid: entity work.
                            hysteresis_spread_elect_element(
                            behaviour_hysteresis_spread_elect_element
                            )
93                              port map(pixel_in => image_in(i,j),
94                                       pixel_n => image_in(i-1,j),
95                                       pixel_s => NOT_CONTOUR,
96                                       pixel_w => image_in(i,j-1),
97                                       pixel_e => image_in(i,j+1),
98                                       pixel_nw => image_in(i-1,j
                                            -1),
99                                       pixel_ne => image_in(i-1,j
                                            +1),
100                                      pixel_se => NOT_CONTOUR,
101                                      pixel_sw => NOT_CONTOUR,
102                                      reset => reset,
103                                      clk => clk,
104                                      pixel_out => image_out(i,j)
                                            );
105                 end generate south;
106
107                 south_east: if i = MAX_LIN - 1 and j = MAX_COL - 1
                        generate
108                     grid: entity work.
                            hysteresis_spread_elect_element(
                            behaviour_hysteresis_spread_elect_element
                            )
109                             port map(pixel_in => image_in(i,j),
110                                      pixel_n => image_in(i-1,j),
111                                      pixel_s => NOT_CONTOUR,
112                                      pixel_w => image_in(i,j-1),
113                                      pixel_e => NOT_CONTOUR,
114                                      pixel_nw => image_in(i-1,j
                                            -1),
115                                      pixel_ne => NOT_CONTOUR,
116                                      pixel_se => NOT_CONTOUR,
117                                      pixel_sw => NOT_CONTOUR,
118                                      reset => reset,
119                                      clk => clk,
120                                      pixel_out => image_out(i,j)
                                            );
121                 end generate south_east;
122
123                 west: if i > 0 and i < MAX_LIN - 1 and j = 0
                        generate
```

```
124                         grid: entity work.
                                hysteresis_spread_elect_element(
                                behaviour_hysteresis_spread_elect_element
                                )
125                                 port map(pixel_in => image_in(i,j),
126                                         pixel_n => image_in(i-1,j),
127                                         pixel_s => image_in(i+1,j),
128                                         pixel_w => NOT_CONTOUR,
129                                         pixel_e => image_in(i,j+1),
130                                         pixel_nw => NOT_CONTOUR,
131                                         pixel_ne => image_in(i-1,j
                                            +1),
132                                         pixel_se => image_in(i+1,j
                                            +1),
133                                         pixel_sw => NOT_CONTOUR,
134                                         reset => reset,
135                                         clk => clk,
136                                         pixel_out => image_out(i,j)
                                            );
137                     end generate west;
138
139                     east: if i > 0 and i < MAX_LIN - 1 and j = MAX_COL
                            - 1 generate
140                         grid: entity work.
                                hysteresis_spread_elect_element(
                                behaviour_hysteresis_spread_elect_element
                                )
141                                 port map(pixel_in => image_in(i,j),
142                                         pixel_n => image_in(i-1,j),
143                                         pixel_s => image_in(i+1,j),
144                                         pixel_w => image_in(i,j-1),
145                                         pixel_e => NOT_CONTOUR,
146                                         pixel_nw => image_in(i-1,j
                                            -1),
147                                         pixel_ne => NOT_CONTOUR,
148                                         pixel_se => NOT_CONTOUR,
149                                         pixel_sw => image_in(i+1,j
                                            -1),
150                                         reset => reset,
151                                         clk => clk,
152                                         pixel_out => image_out(i,j)
                                            );
153                     end generate east;
154
155                     center: if i > 0 and i < MAX_LIN - 1 and j > 0 and
                            j < MAX_COL - 1 generate
```

```
156                                 grid: entity work.
                                        hysteresis_spread_elect_element(
                                        behaviour_hysteresis_spread_elect_element
                                        )
157                                         port map(pixel_in => image_in(i,j),
158                                                 pixel_n => image_in(i-1,j),
159                                                 pixel_s => image_in(i+1,j),
160                                                 pixel_w => image_in(i,j-1),
161                                                 pixel_e => image_in(i,j+1),
162                                                 pixel_nw => image_in(i-1,j
                                                    -1),
163                                                 pixel_ne => image_in(i-1,j
                                                    +1),
164                                                 pixel_se => image_in(i+1,j
                                                    +1),
165                                                 pixel_sw => image_in(i+1,j
                                                    -1),
166                                                 reset => reset,
167                                                 clk => clk,
168                                                 pixel_out => image_out(i,j)
                                                    );
169                             end generate center;
170
171                     end generate columns;
172             end generate lines;
173
174 end behaviour_hysteresis_spread_elect;
```

E.10 Código-fonte hysteresis_spread_elect_element.vhdl

```
1  use work.work_type.all;
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  ------------------------------------
8  -------- Entity Declaration --------
9  ------------------------------------
10
11 entity hysteresis_spread_elect_element is
12         port(
13                 pixel_in:in uint2;
14                 pixel_n:in uint2;
15                 pixel_s:in uint2;
```

```
16                  pixel_w:in uint2;
17                  pixel_e:in uint2;
18                  pixel_nw:in uint2;
19                  pixel_ne:in uint2;
20                  pixel_se:in uint2;
21                  pixel_sw:in uint2;
22                  reset:in bit;
23                  clk:in bit;
24                  pixel_out: out uint2
25          );
26  end hysteresis_spread_elect_element;
27
28  ---------------------------------------------
29  ------------- Architecture ------------------
30  ---------------------------------------------
31
32  architecture behaviour_hysteresis_spread_elect_element of
        hysteresis_spread_elect_element is
33  signal buf: uint2;
34  begin
35
36          behav: process(reset,clk) is
37          variable sum: uint2;
38          begin
39
40
41                  if reset = '1' then
42                          buf <= (others => '0');
43                  elsif clk'event and clk = '1' then
44                          -- Truque aqui
45                          -- Eh feito o somatorio de todos os vizinhos.
46                          sum :=  ((pixel_n or pixel_s) or
47                                   (pixel_w or pixel_e)) or
48                                  ((pixel_nw or pixel_ne) or
49                                  (pixel_se or pixel_sw));
50
51                          if pixel_in = CANDIDATE then
52
53                                  if sum = ALL_NOT_CONTOUR then
54                                          buf <= NOT_CONTOUR;
55                                  elsif sum > ALL_CANDIDATES then
56                                          buf <= CONTOUR;
57                                  else
58                                          buf <= CANDIDATE;
59                                  end if;
60                          else
61                                  buf <= pixel_in;
```

```
62                          end if;
63
64                  end if;
65
66          end process behav;
67
68          pixel_out <= buf;
69
70
71  end behaviour_hysteresis_spread_elect_element;
```

E.11 Código-fonte hysteresis_sum_element.vhdl

```
1   use work.work_type.all;
2
3   ---------------------------------------
4   -------- Entity Declaration ----------
5   ---------------------------------------
6
7   entity hysteresis_sum_element is
8           port(
9                   bit_00:in bit;
10                  bit_01:in bit;
11                  bit_02:in bit;
12                  bit_03:in bit;
13                  bit_04:in bit;
14                  bit_05:in bit;
15                  bit_06:in bit;
16                  bit_07:in bit;
17                  bit_08:in bit;
18                  bit_09:in bit;
19                  bit_10:in bit;
20                  bit_11:in bit;
21                  bit_12:in bit;
22                  bit_13:in bit;
23                  bit_14:in bit;
24                  bit_15:in bit;
25                  reset:in bit;
26                  clk:in bit;
27                  bit_out: out bit
28          );
29  end hysteresis_sum_element;
30
31  -----------------------------------------------
32  ---------------- Architecture -----------------
```

```
33 ----------------------------------------------
34
35 architecture behaviour_hysteresis_sum_element of hysteresis_sum_element is
36 signal buf: bit;
37 begin
38
39         behav: process(reset,clk) is
40         begin
41                 if reset = '1' then
42                         buf <= '0';
43                 elsif clk'event and clk = '1' then
44
45                         buf <=  (((bit_00 or bit_01) or (bit_02 or bit_03))
                                or
46                                 ((bit_04 or bit_05) or (bit_06 or bit_07)))
                                    or
47                                 (((bit_08 or bit_09) or (bit_10 or bit_11))
                                    or
48                                 ((bit_12 or bit_13) or (bit_14 or bit_15)))
                                    ;
49
50                 end if;
51
52                 bit_out <= buf;
53
54         end process behav;
55
56
57
58
59 end behaviour_hysteresis_sum_element;
```

E.12 Código-fonte hysteresis_sum_first.vhdl

```
1  use work.work_type.all;
2
3  ---------------------------------------
4  -------- Entity Declaration ---------
5  ---------------------------------------
6
7  entity hysteresis_sum_first is
8          port(
9                  bit_array_in:in bit_array;
10                 reset:in bit;
11                 clk:in bit;
```

```
12                  bit_array_out:out bit_array_div_4
13          );
14  end hysteresis_sum_first;
15
16  ------------------------------------------
17  ------------- Architecture ---------------
18  ------------------------------------------
19
20  architecture behaviour_hysteresis_sum_first of hysteresis_sum_first is
21  begin
22
23          lines: for lin in 0 to MAX_LIN/4 - 1 generate
24          constant i: natural:= lin * 4;
25          begin
26                  columns: for col in 0 to MAX_COL/4 - 1 generate
27                  constant j: natural:= col * 4;
28                  begin
29                          grid: entity work.hysteresis_sum_element(
                                behaviour_hysteresis_sum_element)
30                                  port map(bit_00 => bit_array_in(i,j),
31                                          bit_01 => bit_array_in(i,j+1),
32                                          bit_02 => bit_array_in(i,j+2),
33                                          bit_03 => bit_array_in(i,j+3),
34                                          bit_04 => bit_array_in(i+1,j),
35                                          bit_05 => bit_array_in(i+1,j+1),
36                                          bit_06 => bit_array_in(i+1,j+2),
37                                          bit_07 => bit_array_in(i+1,j+3),
38                                          bit_08 => bit_array_in(i+2,j),
39                                          bit_09 => bit_array_in(i+2,j+1),
40                                          bit_10 => bit_array_in(i+2,j+2),
41                                          bit_11 => bit_array_in(i+2,j+3),
42                                          bit_12 => bit_array_in(i+3,j),
43                                          bit_13 => bit_array_in(i+3,j+1),
44                                          bit_14 => bit_array_in(i+3,j+2),
45                                          bit_15 => bit_array_in(i+3,j+3),
46                                          reset => reset,
47                                          clk => clk,
48                                          bit_out => bit_array_out(lin,col));
49
50                  end generate columns;
51          end generate lines;
52
53  end behaviour_hysteresis_sum_first;
```

E.13 Código-fonte hysteresis_sum_second.vhdl

```
1  use work.work_type.all;
2
3  ----------------------------------------
4  -------- Entity Declaration ------------
5  ----------------------------------------
6
7  entity hysteresis_sum_second is
8          port(
9                  bit_array_in:in bit_array_div_4;
10                 reset:in bit;
11                 clk:in bit;
12                 bit_array_out:out bit_array_div_16
13         );
14 end hysteresis_sum_second;
15
16 ------------------------------------------------
17 ---------------- Architecture ------------------
18 ------------------------------------------------
19
20 architecture behaviour_hysteresis_sum_second of hysteresis_sum_second is
21 begin
22
23         lines: for lin in 0 to MAX_LIN/16 - 1 generate
24         constant i: natural:= lin * 4;
25         begin
26                 columns: for col in 0 to MAX_COL/16 - 1 generate
27                 constant j: natural:= col * 4;
28                 begin
29                         grid: entity work.hysteresis_sum_element(
                               behaviour_hysteresis_sum_element)
30                             port map(bit_00 => bit_array_in(i,j),
31                                     bit_01 => bit_array_in(i,j+1),
32                                     bit_02 => bit_array_in(i,j+2),
33                                     bit_03 => bit_array_in(i,j+3),
34                                     bit_04 => bit_array_in(i+1,j),
35                                     bit_05 => bit_array_in(i+1,j+1),
36                                     bit_06 => bit_array_in(i+1,j+2),
37                                     bit_07 => bit_array_in(i+1,j+3),
38                                     bit_08 => bit_array_in(i+2,j),
39                                     bit_09 => bit_array_in(i+2,j+1),
40                                     bit_10 => bit_array_in(i+2,j+2),
41                                     bit_11 => bit_array_in(i+2,j+3),
42                                     bit_12 => bit_array_in(i+3,j),
43                                     bit_13 => bit_array_in(i+3,j+1),
```

```
44                                          bit_14 => bit_array_in(i+3,j+2),
45                                          bit_15 => bit_array_in(i+3,j+3),
46                                          reset => reset,
47                                          clk => clk,
48                                          bit_out => bit_array_out(lin,col));
49
50              end generate columns;
51      end generate lines;
52
53  end behaviour_hysteresis_sum_second;
```

E.14 Código-fonte hysteresis_sum_third.vhdl

```
1   use work.work_type.all;
2
3   ----------------------------------
4   -------- Entity Declaration --------
5   ----------------------------------
6
7   entity hysteresis_sum_third is
8           port(
9                   bit_array_in:in bit_array_div_16;
10                  reset:in bit;
11                  clk:in bit;
12                  res:out bit
13          );
14  end hysteresis_sum_third;
15
16  ------------------------------------------
17  ------------ Architecture ------------
18  ------------------------------------------
19
20  architecture behaviour_hysteresis_sum_third of hysteresis_sum_third is
21  begin
22
23          grid: entity work.hysteresis_sum_element(
                behaviour_hysteresis_sum_element)
24                  port map(bit_00 => bit_array_in(0,0),
25                          bit_01 => bit_array_in(0,1),
26                          bit_02 => bit_array_in(0,2),
27                          bit_03 => bit_array_in(0,3),
28                          bit_04 => bit_array_in(1,0),
29                          bit_05 => bit_array_in(1,1),
30                          bit_06 => bit_array_in(1,2),
31                          bit_07 => bit_array_in(1,3),
```

```
32                          bit_08 => bit_array_in(2,0),
33                          bit_09 => bit_array_in(2,1),
34                          bit_10 => bit_array_in(2,2),
35                          bit_11 => bit_array_in(2,3),
36                          bit_12 => bit_array_in(3,0),
37                          bit_13 => bit_array_in(3,1),
38                          bit_14 => bit_array_in(3,2),
39                          bit_15 => bit_array_in(3,3),
40                          reset => reset,
41                          clk => clk,
42                          bit_out => res);
43
44 end behaviour_hysteresis_sum_third;
```

E.15 Código-fonte mux.vhdl

```
1  use work.work_type.all;
2
3  ---------------------------------------
4  -------- Entity Declaration ---------
5  ---------------------------------------
6
7  entity mux is
8          port(
9                  image_in_a:in image_u2;
10                 image_in_b:in image_u2;
11                 sel:in bit;
12                 reset:in bit;
13                 clk:in bit;
14                 image_out:out image_u2
15         );
16 end mux;
17
18 -----------------------------------------------
19 ------------- Architecture ------------------
20 -----------------------------------------------
21
22 architecture behaviour_mux of mux is
23 begin
24
25         lines: for i in 0 to MAX_LIN - 1 generate
26         begin
27                 columns: for j in 0 to MAX_COL - 1 generate
28                 begin
```

```
29                          grid: entity work.mux_element(behaviour_mux_element
                                )
30                                  port map(a => image_in_a(i,j),
31                                           b => image_in_b(i,j),
32                                           sel => sel,
33                                           reset => reset,
34                                           clk => clk,
35                                           pixel_out => image_out(i,j));
36
37                  end generate columns;
38          end generate lines;
39
40  end behaviour_mux;
```

E.16 Código-fonte mux_element.vhdl

```
1   use work.work_type.all;
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.numeric_std.all;
6
7   ------------------------------------
8   -------- Entity Declaration --------
9   ------------------------------------
10
11  entity mux_element is
12          port(
13                  a: uint2;
14                  b: uint2;
15                  sel:in bit;
16                  reset:in bit;
17                  clk:in bit;
18                  pixel_out:out uint2
19          );
20  end mux_element;
21
22  architecture behaviour_mux_element of mux_element is
23  signal counter: integer_u8;
24  signal buf: uint2;
25  begin
26
27          behav : process(reset,clk) is
28          begin
29
```

```
30                  if reset = '1' then
31                          buf <= (others => '0');
32                          counter <= 0;
33                  elsif clk'event and clk = '1' then
34
35                          if counter < COUNTER_LIMIT then
36                                  buf <= a;
37                                  counter <= counter + 1;
38                          else
39                                  if sel = '0' then
40                                          buf <= a;
41                                  else
42                                          buf <= b;
43                                  end if;
44
45                          end if;
46
47                  end if;
48
49                  pixel_out <= buf;
50
51          end process behav;
52
53
54  end behaviour_mux_element;
```

E.17 Código-fonte mux_tag.vhdl

```
1   use work.work_type.all;
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.numeric_std.all;
6
7   ------------------------------------
8   -------- Entity Declaration --------
9   ------------------------------------
10
11  entity mux_tag is
12          port(
13                  a: uint8;
14                  b: uint8;
15                  sel:in bit;
16                  reset:in bit;
17                  clk:in bit;
```

```
18                  pixel_out:out uint8
19          );
20  end mux_tag;
21
22  architecture behaviour_mux_tag of mux_tag is
23  signal counter: integer_u8;
24  signal buf: uint8;
25  begin
26
27          behav : process(reset,clk) is
28
29          begin
30
31                  if reset = '1' then
32                          buf <= (others => '0');
33                          counter <= 0;
34                  elsif clk'event and clk = '1' then
35
36                          if counter < COUNTER_LIMIT then
37                                  buf <= a;
38                                  counter <= counter + 1;
39                          else
40                                  if sel = '0' then
41                                          buf <= a;
42                                  else
43                                          buf <= b;
44                                  end if;
45
46                          end if;
47
48                  end if;
49
50                  pixel_out <= buf;
51
52          end process behav;
53
54
55
56
57  end behaviour_mux_tag;
```

E.18 Código-fonte parallel_to_serial.vhdl

```
1  use work.work_type.all;
2
```

```
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  ------------------------------------
8  -------- Entity Declaration --------
9  ------------------------------------
10
11 entity parallel_to_serial is
12         port(
13                 image_in:in image_u8;
14                 tag_in: in uint8;
15                 -- Adaptacao para o hardware da xilinx
16                 address_in:in uint16;
17                 --
18                 retain:in bit;
19                 reset:in bit;
20                 clk:in bit;
21                 ready:out bit;
22                 finish:out bit;
23                 tag_out: out uint8;
24                 pixel_out: out uint8
25         );
26 end parallel_to_serial;
27
28 ---------------------------------------------
29 ------------- Architecture ------------------
30 ---------------------------------------------
31
32 architecture behaviour_parallel_to_serial of parallel_to_serial is
33         signal image_buf: image_u8;
34         signal pixel_buf: uint8;
35         signal tag_buf: uint8;
36         signal start: bit;
37 begin
38
39         behav: process(reset, clk) is
40         variable i,j: integer;
41         begin
42                 ------------------------------------------------
43                 -- Signals explanation -------------------------
44                 -- ready  = report first pixel (when '1') ----
45                 -- finish = report last  pixel (when '1') ----
46                 -- start  = an internal control signal    ----
47                 ------------------------------------------------
48
49                 if reset = '1' then
```

```
50                          i := 0;
51                          j := 0;
52                          ready <= '0';
53                          finish <= '0';
54                          start <= '0';
55                      elsif clk'event and clk = '1' then
56
57                          --
58                          if retain = '0' and start = '0' then
59                                  image_buf <= image_in;
60                                  tag_buf <= tag_in;
61
62                                  start <= '1';
63                                  ready <= '0';
64
65                                  i := 0;
66                                  j := 0;
67                          end if;
68
69                          --
70                          if start = '1' then
71
72                                  --
73  --                              if j < MAX_COL and i < MAX_LIN then
74  --                                      pixel_out <= image_buf(i,j);
75  --                              end if;
76
77                                  -- Counting cells
78                                  if j < MAX_COL - 1 then
79                                          j := j + 1;
80                                  else
81                                          i := i + 1;
82                                          j := 0;
83                                  end if;
84
85                                  -- End of buffer verification
86                                  if (i = MAX_LIN - 1) and (j = MAX_COL - 1)
                                        then
87                                          finish <= '1';
88                                  elsif i < MAX_LIN then
89                                          ready <= '0';
90                                  else
91                                          ready <= '1';
92                                          start <= '0';
93                                          finish <= '0';
94                                  end if;
95
```

```
96                          end if;
97
98                  end if;
99
100         end process behav;
101
102
103         behav_output: process(reset, clk) is
104         --behav_output: process(address_in) is
105         variable i,j: integer;
106         begin
107                 ------------------------------------------------
108                 ------------------------------------------------
109
110                 if reset = '1' then
111                         i := 0;
112                         j := 0;
113                 elsif clk'event and clk = '1' then
114
115                         i := to_integer(address_in) / MAX_LIN;
116                         j := to_integer(address_in) rem MAX_COL;
117
118                         if j < MAX_COL and i < MAX_LIN then
119                                 pixel_buf <= image_buf(i,j);
120                         end if;
121
122                 end if;
123
124         end process behav_output;
125
126         pixel_out <= pixel_buf;
127         tag_out   <= tag_buf;
128
129 end behaviour_parallel_to_serial;
```

E.19 Código-fonte serial_to_parallel.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  use work.work_type.all;
6
7  ---------------------------------
8  ------- Entity Declaration -------
```

```
9  ----------------------------------
10
11 entity serial_to_parallel is
12         port(
13                 pixel_in:in uint8;
14                 address_in: uint16;
15                 reset:in bit;
16                 clk:in bit;
17                 ready:out bit;
18                 image_out: out image_u2
19         );
20 end serial_to_parallel;
21
22 ----------------------------------------
23 ------------- Architecture -------------
24 ----------------------------------------
25
26 architecture behaviour_serial_to_parallel of serial_to_parallel is
27 signal ready_buf: bit;
28 begin
29
30         behav: process(reset, clk) is
31
32         variable i,j: integer;
33         begin
34                 if reset = '1' then
35                         i := 0;
36                         j := 0;
37                         ready_buf <= '0';
38                 elsif clk'event and clk = '1' then
39
40                         --
41 --                      if j < MAX_COL and i < MAX_LIN then
42 --                              image_buf(i,j) := pixel_in(1 downto 0);
43 --                      end if;
44
45                         -- Counting cells
46                         if j < MAX_COL - 1 then
47                                 j := j + 1;
48                         else
49                                 i := i + 1;
50                                 j := 0;
51
52                         end if;
53
54                         -- End of buffer verification
55                         if i < MAX_LIN then
```

```
56                                  ready_buf <= '0';
57                          else
58                                  ready_buf <= '1';
59                                  i := 0;
60                          end if;
61
62                  end if;
63
64
65
66          end process behav;
67
68
69          behav_output: process(reset, clk) is
70          variable i,j: integer;
71          begin
72                  ------------------------------------------------
73                  ------------------------------------------------
74
75                  if reset = '1' then
76                          i := 0;
77                          j := 0;
78                  elsif clk'event and clk = '1' then
79
80                          i := to_integer(address_in) / MAX_LIN;
81                          j := to_integer(address_in) rem MAX_COL;
82
83                          if j < MAX_COL and i < MAX_LIN then
84                                  image_out(i,j) <= pixel_in(1 downto 0);
85                          end if;
86
87                  end if;
88
89          end process behav_output;
90
91          ready <= ready_buf;
92
93
94  end behaviour_serial_to_parallel;
```

E.20 Código-fonte work_type.vhdl

```
1  library ieee;
2  use std.textio.all;
3  use ieee.std_logic_1164.all;
```

```
4  use ieee.numeric_std.all;
5
6  package work_type is
7          subtype uint2  is unsigned( 1 downto 0);
8          subtype uint8  is unsigned( 7 downto 0);
9          subtype uint11 is unsigned(10 downto 0);
10         subtype uint16 is unsigned(15 downto 0);
11
12         subtype integer_u8 is integer range 0 to 255;
13
14         constant RANGE_UINT8: uint8 := x"FF"; -- 255
15         constant MAX_LIN: integer := 64;
16         constant MAX_COL: integer := 64;
17         constant NUM_IMAGES: integer := 6;
18         constant MAX_PIX: integer := MAX_LIN * MAX_COL;
19
20         type image_u2 is array (0 to MAX_LIN - 1,0 to MAX_COL - 1) of uint2
               ;
21         type image_u8 is array (0 to MAX_LIN - 1,0 to MAX_COL - 1) of uint8
               ;
22         type bit_array is array (0 to MAX_LIN - 1,0 to MAX_COL - 1) of bit;
23         type bit_array_div_4  is array (0 to MAX_LIN/4  - 1,0 to MAX_COL/4
               - 1) of bit;
24         type bit_array_div_16 is array (0 to MAX_LIN/16 - 1,0 to MAX_COL/16
               - 1) of bit;
25         type test_array is array(1 to 11) of bit;
26         type test_array_int is array(1 to 11) of integer;
27         constant NOT_CONTOUR: uint2 := b"00";
28         constant CANDIDATE: uint2 := b"01";
29         constant CONTOUR: uint2 := b"10";
30         constant ALL_NOT_CONTOUR: uint2 := b"00";
31         constant ALL_CANDIDATES: uint2 := b"01";
32         constant COUNTER_LIMIT: integer := NUM_IMAGES - 1; -- Imagem com
               256x256, COUNTER_LIMIT = 12
33         constant NO_RETAIN: bit := '0';
34
35         procedure print(str: string);
36 end package work_type;
37
38 package body work_type is
39
40         -- Procedure made for debugging
41         procedure print(str: string) is
42         variable output_line : line;
43         begin
44                 write(output_line, str);
45                 writeline(output, output_line);
```

```
46          end print;
47
48  end package body work_type;
```