UNIVERSIDADE FEDERAL DA BAHIA

ESCOLA POLITÉCNICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Lauê Rami Souza Costa de Jesus

IMPLEMENTAÇÃO DE UM SISTEMA DE PROCESSAMENTO EM TEMPO REAL PARA A DETECÇÃO DE OBJETOS EM UMA CENA FIXA EM VÍDEOS DIGITAIS USANDO FPGA

DISSERTAÇÃO DE MESTRADO

Salvador

Outubro de 2020

Página em branco

Lauê Rami Souza Costa de Jesus

IMPLEMENTAÇÃO DE UM SISTEMA DE PROCESSAMENTO EM TEMPO REAL PARA A DETECÇÃO DE OBJETOS EM UMA CENA FIXA EM VÍDEOS DIGITAIS USANDO FPGA

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, PPGEE, da Universidade Federal da Bahia, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientadores: Wagner L. A. de Oliveira
Paulo C. M. de Abreu Farias

Salvador

Outubro de 2020

Ficha catalográfica elaborada pelo Sistema Universitário de Bibliotecas (SIBI/UFBA),
com os dados fornecidos pelo(a) autor(a).

Souza Costa de Jesus, Lauê Rami
Implementação de um Sistema de Processamento em Tempo Real para a Detecção de Objetos em uma Cena Fixa em Vídeos Digitais usando FPGA / Lauê Rami Souza Costa de Jesus. -- Salvador, 2021.
126 f. : il

Orientador: Wagner Luiz Alves de Oliveira.
Coorientador: Paulo Cesar Machado de Abreu Farias.
Dissertação (Mestrado - Programa de Pós-Graduação em Engenharia Elétrica) -- Universidade Federal da Bahia, Escola Politécnica, 2021.

1. Processamento Digital de Imagens. 2. Microeletrônica. 3. Visão Computacional. 4. Detecção de Objetos. 5. FPGA. I. Luiz Alves de Oliveira, Wagner. II. Machado de Abreu Farias, Paulo Cesar. III. Título.

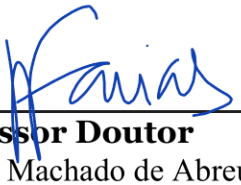
Lauê Rami Souza Costa de Jesus

Implementação de um Sistema de Processamento em Tempo Real para a Detecção de Objetos em uma Cena Fixa em Vídeos Digitais usando FPGA

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal da Bahia como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Trabalho aprovado. Salvador, 27 de novembro de 2020:

Professor Doutor
Wagner Luiz Alves de Oliveira (Orientador)



Professor Doutor
Paulo César Machado de Abreu Farias (Coorientador)

Professor Doutor
Acbal Rucas Andrade Achy
Universidade Federal do Recôncavo da Bahia

Professor Doutor
Edson Pinto Santana
Universidade Federal da Bahia

Salvador
novembro, 2020

Agradecimentos

Agradeço, primeiramente, aos meus pais, pelo irrestrito esforço para me proporcionar uma educação de qualidade.

À minha esposa Ana Caroline Correia de Almeida.

Aos meus orientadores, Prof. Dr. Wagner L. Alves de Oliveira e Prof. Dr. Paulo Cesar.

Resumo da Dissertação apresentada à PPGEE/UFBA como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica (M.Sc.)

IMPLEMENTAÇÃO DE UM SISTEMA DE PROCESSAMENTO EM TEMPO REAL PARA A DETECÇÃO DE OBJETOS EM UMA CENA FIXA EM VÍDEOS DIGITAIS USANDO FPGA

Lauê Rami Souza Costa de Jesus

Outubro/2020

Orientadores: Wagner L. A. de Oliveira
Paulo C. M. de Abreu Farias

Departamento: Engenharia Elétrica e de Computação

O processamento de vídeos e imagens digitais é uma área que possui muitas aplicações que precisam de desempenho em tempo real, por exemplo, vigilância por vídeo e segurança automotiva. Uma aplicação de processamento de imagens em tempo real visa manipular, analisar e processar uma grande quantidade de dados relacionados aos *pixels* de uma imagem em um tempo estipulado. Essas aplicações incluem técnicas de processamento de imagens como análise, aprimoramento e segmentação, bem como a detecção de objetos e outras. A extração das informações de uma imagem é uma tarefa intensiva com alto custo computacional e através do uso de um dispositivo FPGA é possível obter uma implementação mais adequada.

Este trabalho discute a implementação de um sistema de processamento em tempo real para detecção de objetos em vídeos digitais usando FPGA. Todas as etapas do fluxo de projeto do FPGA foram desenvolvidas. Nesta abordagem foram incluídas as etapas de especificação do projeto, modelagem do sistema, microarquitetura, codificação RTL, verificação funcional, síntese lógica, simulação em nível de porta (lógica pós-síntese), análise de tempo e programação de dispositivos FPGA.

A arquitetura de hardware proposta foi projetada e sintetizada para o kit FPGA Altera DE2-115. Os resultados são verificados em tempo real com um vídeo de entrada advindo do sensor Terasic CMOS TRDB-D5M, do dispositivo FPGA DE2-115 e do monitor VGA para a exibição de imagens. A partir dos testes foram obtidos resultados com um *speedup* de 310x para a implementação do algoritmo de detecção de objetos em FPGA, quando comparada à implementação no software Matlab. Além disso, os resultados de síntese lógica mostraram que a arquitetura de hardware implementada teve uma baixa utilização dos recursos do FPGA, possibilitando que o sistema seja implementado em um dispositivo de baixo custo.

Abstract of Dissertation presented to PPGEE/UFBA as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

IMPLEMENTATION OF A REAL-TIME PROCESSING SYSTEM FOR OBJECT DETECTION IN A FIXED SCENE IN DIGITAL VIDEOS USING FPGA

Lauê Rami Souza Costa de Jesus

October/2020

Advisors: Wagner L. A. de Oliveira
Paulo C. M. de Abreu Farias

Department: Electrical and Computer Engineering

Digital video and image processing is an area that finds many applications that need real-time performance, for example, Video Surveillance and Automotive Safety. A real-time image processing application aims to manipulate, analyze and process a large amount of data related to the pixels of an image in a stipulated time. These applications includes image processing techniques like image analysis, image enhancement, image segmentation, object detection and others. Extracting the resources of an image is an intensive task with high computational cost and through the use of an FPGA device it is possible to achieve a more suitable implementation.

This work discusses the implementation of a Real-Time Processing System for Object Detection in Digital Videos using FPGA. All the steps of the FPGA design flow were developed. In this approach the steps of project specification, system modeling, microarchitecture, RTL coding, functional verification, logic synthesis, gate level simulation (logic post-synthesis), time analysis and FPGA device programming were included.

The proposed hardware architecture was designed and synthesized for Altera DE2-115 FPGA kit. The results are verified in real time with an input video from

TRDB-D5M Terasic CMOS sensor, DE2-115 FPGA kit and VGA monitor for displaying images. The tests results were obtained with a 310x speedup for the object detection algorithm implementation in FPGA, when compared to the Matlab implementation. In addition, the logic synthesis results have shown that the hardware architecture implemented had a low FPGA resources utilization, enabling the system to be developed in a low cost device.

Sumário

Lista de Figuras

Lista de Tabelas

Dedico esse trabalho a minha esposa Ana Caroline Correia de Almeida de Jesus e aos meus pais, Nagilson Costa de Jesus e Maria Zenaide de Souza

Capítulo 1

Introdução

A detecção de objetos é o ato de encontrar um objeto em uma imagem. Hoje, os algoritmos de detecção de objetos são usados em aplicações como a vigilância por vídeo [1] e segurança, rastreamento de objetos, reconhecimento facial [2], inspeção industrial [3], processamento e análise de imagens médicas [4], carros autodirigidos e outros.

As estratégias de visão computacional para detecção de objetos são comumente desenvolvidas em software. Existem diferentes trabalhos como o de [5], [6] e [7] que estudam a implementação de diversos algoritmos de detecção de objetos em software. Nestes trabalhos os algoritmos de processamento de imagem são implementados em Matlab ou OpenCV, no entanto poderiam ser otimizados em termos de desempenho e consumo de potência utilizando outras abordagens para implementação. Muitas técnicas de detecção de objetos usam operadores de detecção de bordas como uma etapa do processamento de dados de imagem. Os métodos de detecção de bordas consistem em análise de imagem e extração de informações estruturais. Essa informação extraída referente à borda reduz a quantidade de dados a ser processada. No entanto, o processo de detecção de bordas é computacionalmente exaustivo, pois para cada pixel operado pelo detector de borda, cerca de 20 operações lógicas e aritméticas são executadas. Além disso, para a realização dessas operações, devem ser consideradas as sobrecargas de armazenamento e recuperação de dados dos *pixels* da imagem. Portanto, a restrição de tempo é o maior desafio neste tipo de aplicação.

O FPGA é uma plataforma reconfigurável, que fornece grandes matrizes de recursos lógicos programáveis. O FPGA também oferece o paralelismo temporal e

espacial, e essa característica torna o FPGA adequado para a implementação de algoritmos de processamento de imagem em tempo real. O aumento na capacidade de dispositivos FPGA, com relação a recursos e desempenho, tem resultado em projetos muito mais complexos, levando ao desenvolvimento de aplicações utilizando o fluxo de projeto em FPGA. De acordo com [8], a utilização de arquiteturas de hardware reconfigurável tem crescido entre os projetistas de diferentes áreas como robótica, telecomunicações, processamento digital de sinais e imagens, equipamentos médicos e automotiva, bem como na área de projetos de circuitos integrados. Segundo [9], foram obtidos avanços com a utilização de dispositivos FPGA para a aceleração da computação de alto desempenho.

O uso de dispositivos FPGA está aumentando à medida que fornecem um baixo tempo de colocação no mercado e custo mínimo de desenvolvimento de hardware, recursos muito procurados pela rápida mudança tecnologias. Além disso, o processo de verificação funcional contido no fluxo de projeto em FPGA permite maior robustez ao projeto, além de diminuir o tempo gasto para encontrar falhas. No entanto, essa melhora na produtividade do projeto somente será alcançada com a aplicação de uma metodologia de projeto adequada.

1.1 Objetivos

O principal objetivo deste trabalho é desenvolver uma arquitetura de hardware, para a detecção de objetos em tempo real, e prototipar a solução desenvolvida em FPGA.

Para isso, foram utilizados conceitos e técnicas de processamento de imagens, como segmentação e detecção de bordas. O desenvolvimento dessa solução levou em consideração a aplicação do fluxo de projeto de hardware digital, desde a especificação do sistema até a geração do *bitstream*, para a prototipação em FPGA.

Como objetivos específicos desse projeto, cita-se:

1. Estudar conceitos e técnicas de processamento digital de imagens relacionados à segmentação e detecção de bordas.

2. Compreender o fluxo de projeto de hardware digital para utilização como metodologia de implementação.

3. Especificar o sistema de detecção de objetos em uma cena fixa em vídeos digitais.

4. Modelar o sistema de detecção de objetos em uma cena fixa em vídeos digitais em uma linguagem de alto nível.

5. Descrever o hardware de forma a atender os requisitos da especificação funcional.

6. Desenvolver a verificação funcional visando garantir o funcionamento do hardware.

1.2 Organização da Dissertação

No Capítulo 2, são apresentados os conceitos e técnicas relacionadas ao processamento digital de imagens e vídeos. Estes são conceitos fundamentais para o entendimento da aplicação desse projeto.

O Capítulo 3, apresenta a metodologia de desenvolvimento deste projeto, descrevendo todas as etapas do fluxo de projeto de hardware digital (especificação e modelagem do sistema, microarquitetura, codificação RTL, verificação funcional, síntese lógica, síntese física, simulação em nível de portas e geração do *bitstream*. As principais técnicas e ferramentas utilizadas em cada estágio serão detalhadas neste capítulo.

No Capítulo 4, são apresentados os projetos de arquitetura dos principais algoritmos desenvolvidos e utilizados neste trabalho. O fluxo de execução do algoritmo, análise de outras soluções e as técnicas de implementação, que proporcionaram a elaboração dos algoritmos de detecção de bordas e de objetos, também são discutidos.

No Capítulo 5, são apresentados a arquitetura de hardware desenvolvida, a descrição da funcionalidade, diagramas e a lógica de controle de cada bloco implementado em código RTL. No Capítulo 7 é realizada uma análise sobre a restrição de tempo real empregada no projeto, além da discussão de resultados da simulação funcional, síntese lógica, desempenho e prototipação em FPGA, relativos ao sistema de detecção de objetos em uma cena fixa. Os resultados da verificação funcional,

bem como da cobertura funcional também são reportados.

No Capítulo 8 são apresentadas as conclusões deste trabalho. Por fim, no apêndice A encontra-se o texto completo do artigo publicado e apresentado no WCAS 2019.

1.3 Resultados em Publicações

A partir do desenvolvimento, implementação e prototipação em FPGA da arquitetura de hardware, para a detecção de objetos em tempo real, foram obtidos resultados que foram apresentados e publicados no *Workshop on Circuits and System Design* (WCAS 2019), no evento internacional de microeletrônica Chip-in-Sampa 2019, sediado no Brasil.

Abstract: This paper discusses the implementation of a Real-Time Processing System for Object Detection in Digital Videos. The system consists of real-time image processing and this application is suitable for implementation in FPGA due to the algorithm features and the device resources. The proposed hardware architecture was designed and synthesized for Intel DE2-115 FPGA. The main concepts, techniques, the methodology used for development the design will be presented. The results are verified in real time with an input video from TRDB-D5M Terasic CMOS sensor, DE2-115 FPGA device and VGA monitor for displaying images.

JESUS, L. R. S. C., OLIVEIRA, W. L. A., FARIAS, P. C. M. A. **Implementation of a Real-Time Processing System for Object Detection in Digital Videos using FPGA**. *Workshop on Circuits and System Design* (WCAS 2019), São Paulo, Brazil. O Artigo completo publicado no WCAS 2019 está anexado à dissertação.

Capítulo 2

Processamento Digital de Imagens e Vídeos

2.1 Etapas Fundamentais no Processamento Digital de Imagens

Como pode ser visualizado na Figura 2.1, as etapas fundamentais no Processamento Digital de Imagens são organizadas da seguinte forma [10]:

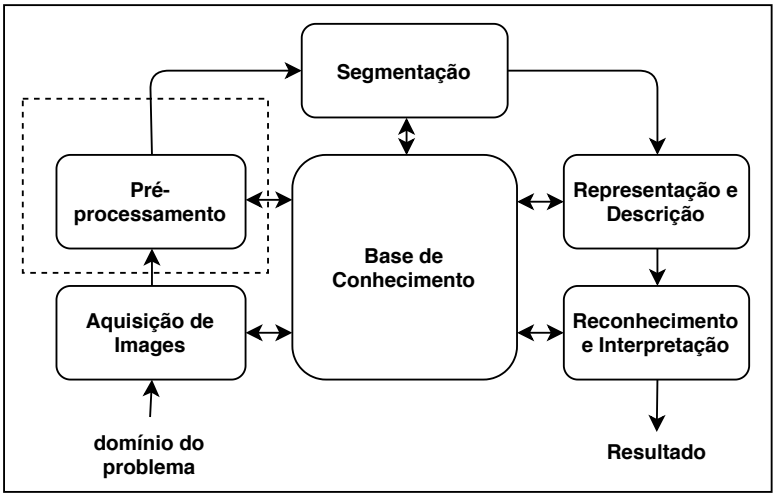


Figura 2.1: Etapas fundamentais no processamento digital de imagens [10].

- Aquisição de Imagens. Um dispositivo físico, sensível a uma banda do espectro eletromagnético, produzirá um sinal elétrico como saída proporcional ao nível de energia percebida. Em seguida, um digitalizador realizará a conversão da

saída elétrica do dispositivo físico para a forma digital.

- Pré-processamento. Este processamento pode incluir etapas distintas como, por exemplo: Melhoria da Imagem, Restauração da Imagem, Processamento de Imagem Colorida, *Wavelets* e processamento multi-resolução, Compressão e Processamento Morfológico. Em resumo, técnicas de processamento de imagens são aplicadas visando a melhoria da imagem, para extração de informações.

- Segmentação. Os procedimentos de segmentação dividem uma imagem em seus componentes ou objetos. A segmentação é uma das etapas mais complexas no processamento de imagens digital. Quanto melhor for o algoritmo de segmentação, maior a chance de sucesso no reconhecimento da imagem.

- Representação e Descrição. A escolha de uma representação é apenas parte da solução, para transformar dados brutos em uma forma adequada para o processamento subsequente do computador. Um método deve ser especificado para descrever os dados, de modo que os recursos de interesse sejam destacados.

- Reconhecimento de Objetos. O reconhecimento é o processo que atribui um rótulo a um objeto com base nos seus descritores.

2.2 Imagem

Segundo [10], uma imagem pode ser definida como uma função bidimensional, $f(x, y)$, onde x e y são coordenadas espaciais (planas), e a amplitude de f em qualquer par de coordenadas (x, y), é chamada intensidade ou nível de cinza da imagem nesse ponto. Quando x, y, e os valores de amplitude de f são todos finitos e quantidades discretas, tem-se uma imagem digital.

O campo de processamento de imagem digital refere-se ao processamento de imagens por meio de um computador. Uma imagem digital é composta por um número finito de elementos, cada um dos quais com uma localização particular e valor. Estes elementos são referidos como elementos de imagem. *Pixel* é o termo mais utilizado para denotar os elementos de uma imagem digital [10].

A convenção do eixo usada para representar a imagem é mostrada na Figura 2.2. A origem está localizada no canto superior esquerdo. A linha horizontal e a linha vertical na origem são definidas como eixos y e x, respectivamente.

Figura 2.2: Coordenadas, convenção e imagem [11].

2.2.1 Amostragem e Quantização da Imagem

Para criar uma imagem digital, é preciso converter os dados sensíveis contínuos em formato digital, o que envolve dois processos que são chamados de Amostragem e Quantização [10].

Uma imagem pode ser contínua em relação às coordenadas x e y, bem como em amplitude. Para convertê-la em forma digital, tem-se que amostrar a função em ambas as coordenadas e amplitude. A discretização dos valores das coordenadas é chamada de amostragem. A discretização dos valores das amplitudes é chamada de quantização.

Amostragem é o processo de conversão de um espaço contínuo (ou sinal de espaço/tempo contínuo) em um sinal de espaço discreto (ou espaço/tempo discreto). Ao adquirir a imagem a função contínua é amostrada e sua amplitude quantizada, a Figura 2.3 ilustra o resultado da amostragem de um sinal unidimensional de domínio contínuo.

A quantização pode ser definida como a aproximação do valor obtido pelo processo de amostragem, para o nível mais próximo dentre um número finito de níveis de quantização.

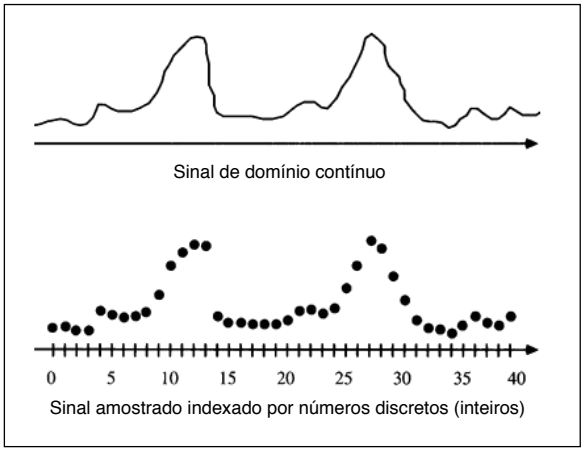


Figura 2.3: Amostragem [12].

No caso de imagens monocromáticas, a função é $f(x, y)$ e os níveis de quantização são também conhecidos como níveis de cinza. É comum adotar N níveis de quantização para digitalização de imagens, onde N é geralmente uma potência de 2, ou seja, $N = 2^n$, sendo n o número de bits necessários para codificar cada valor de *pixel*.

A quantização de imagens pode ser descrita como um processo de mapeamento pelo qual grupos de pontos de dados (vários *pixels* dentro de uma faixa de valores cinza) são mapeados para um único ponto (ou seja, um único nível de cinza). Este processo pode ser visualizado na Figura 2.4. Neste caso, os valores no intervalo de 0 a 255 são representados por apenas 4 níveis de quantização.

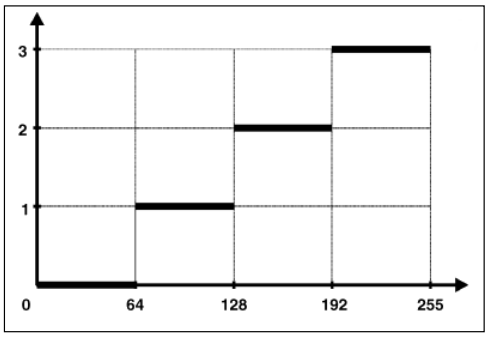


Figura 2.4: Função de mapeamento para quantização uniforme [11].

2.2.2 Representação de Imagens Digitais

Assumindo que uma imagem $f(x, y)$ é amostrada, tem-se que o resultado da imagem digital possui M linhas e N colunas.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix} \tag{2.1}$$

O lado direito da equação 2.1 é por definição uma imagem digital. Cada elemento desta matriz é chamado de elemento de imagem ou *pixel*.

Seja Z e R o conjunto de números inteiros e o conjunto de números reais, respectivamente. O processo de amostragem pode ser visto como um particionamento do plano xy em uma grade, com as coordenadas do centro de cada grade sendo um par de elementos do produto cartesiano Z^2, que é o conjunto de todos os pares ordenados de elementos (z_i, z_j), com z_i e z_j sendo inteiros de Z.

Portanto, $f(x, y)$ é uma imagem digital se (x, y) são inteiros de Z^2, e f é uma função que atribui um valor de intensidade (ou seja, um número que pertence ao conjunto de números reais R) para cada par distinto de coordenadas (x, y) [10]. Se os níveis de cinza também são números inteiros, Z substitui R e uma imagem digital torna-se uma função 2-D cujas coordenadas e valores de amplitude são inteiros.

2.2.3 Vizinhança de um *Pixel*

Um *pixel* p nas coordenadas (x, y) tem quatro vizinhos horizontais e verticais, cujas coordenadas podem ser visualizadas na seguinte forma:

$$(x+1, y), (x-1, y), (x, y+1), (x, y-1) \tag{2.2}$$

Este conjunto de *pixels* é chamado de 4-vizinhança de p. Cada *pixel* está a uma unidade de distância de (x, y), e alguns dos vizinhos de p estão fora da imagem digital se (x, y) estiver na borda da imagem. Os quatro vizinhos que estão na diagonal de p, têm as seguintes coordenadas [13]:

$$(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1) \tag{2.3}$$

Este conjunto de *pixels*, somado ao conjunto citado em 2.2, formam a 8-vizinhança de p. A maioria das vizinhanças usadas em algoritmos de processamento de imagem são matrizes relativamente pequenas, com o número de linhas igual ao número de colunas, e com um número ímpar de *pixels* como, por exemplo, a vizinhança $3x3$ mostrada na Figura 2.5.

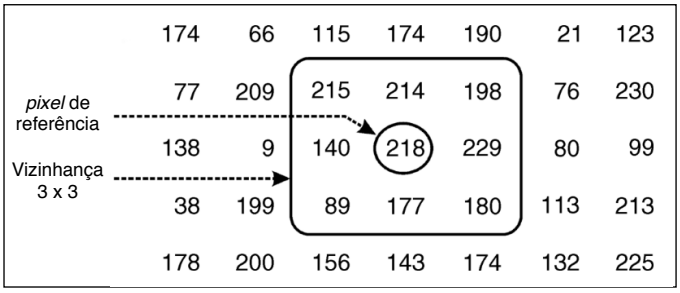


Figura 2.5: *Pixel* dentro de uma vizinhança [11].

2.2.4 Operações Orientadas a Vizinhança

As operações orientadas a vizinhança (também conhecidas como locais ou de área) consistem em determinar o valor de um *pixel* resultante em coordenadas (x, y) como uma função de seu valor original e o valor de seus vizinhos, geralmente usando uma operação de convolução [11]. A convolução entre uma imagem e um *array* 2D (conhecido como janela, máscara ou *kernel*) produz uma imagem de destino, na qual cada valor de *pixel* depende de seu valor original e do valor dos seus vizinhos. A máscara de convolução determina quais vizinhos são usados, bem como o peso relativo de seus valores originais.

Cada coeficiente da máscara $(W_1, ..., W_9)$ pode ser interpretado como um peso. A máscara pode ser considerada como uma pequena janela, a qual é sobreposta na imagem para executar o cálculo em um *pixel* por vez. À medida que cada *pixel* é processado, a janela se move para o próximo *pixel* na imagem, e o processo é repetido até que o último *pixel* tenha sido processado.

A convolução é uma operação matemática fundamental, envolvida em algoritmos de processamento de imagem, orientados para a vizinhança linear. A operação pode ser visualizada na Figura 2.7.

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

Figura 2.6: Máscara de convolução $3x3$.

Fonte: Adaptado de [11].

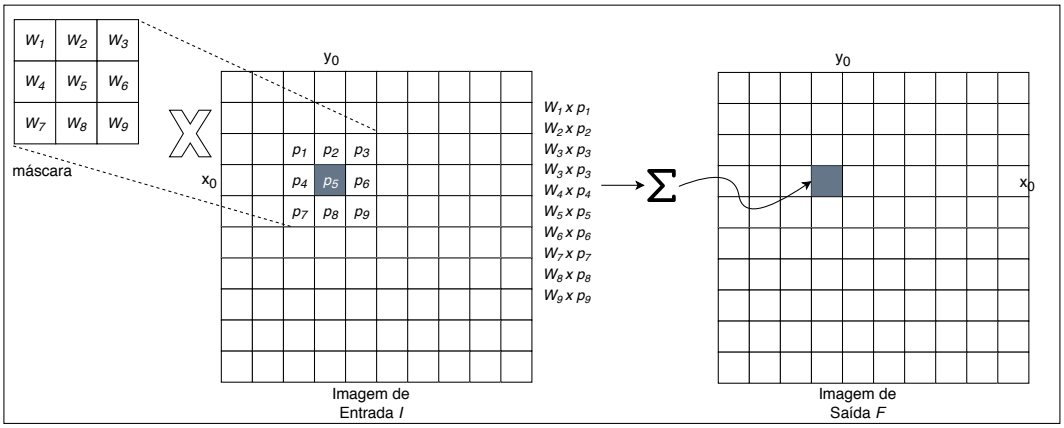


Figura 2.7: Processamento de vizinhança para o caso de filtragem linear.

Fonte: Adaptado de [11].

2.2.5 Tipos de Processamento Digital de Imagens

No campo de processamento de imagem digital, é possível considerar três tipos de processos informatizados: processos de baixo, médio e alto nível. Processos de baixo nível envolvem operações primitivas, como o pré-processamento de imagens para reduzir o ruído, o aumento do contraste e a nitidez da imagem. Um processo de baixo nível é caracterizado pelo fato de que as entradas e as saídas são imagens.

O processamento de nível médio em imagens envolve tarefas como a segmentação, a descrição desses objetos para reduzi-los a uma forma adequada para processamento computacional e a classificação de objetos individuais. Um processo de nível médio é caracterizado, geralmente, pelo fato de que suas saídas são atributos extraídos dessas imagens (por exemplo, arestas, contornos e a identidade dos objetos individuais). Finalmente, o processamento de alto nível envolve um conjunto de objetos reconhecidos, como na interpretação de imagens [10]. A Figura 2.8 apresenta a estrutura dos processos.

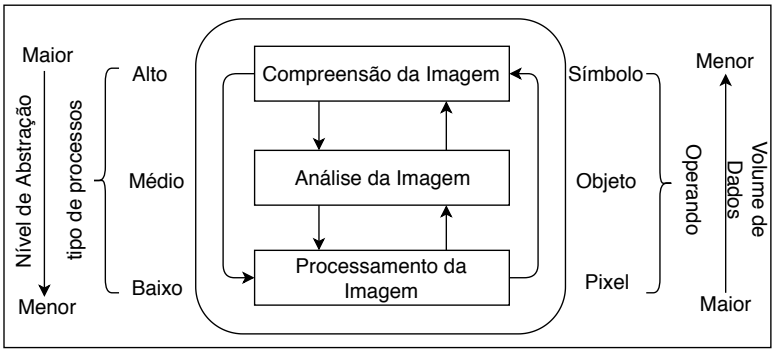


Figura 2.8: Tipos de processamento digital de imagens [14].

2.2.6 Segmentação de Imagem

A Segmentação subdivide uma imagem em suas regiões ou objetos constituintes. A Segmentação deve parar quando os objetos de interesse em uma aplicação forem isolados. Os algoritmos de segmentação de imagens geralmente são baseados em uma das duas propriedades básicas de valores de intensidade: descontinuidade e similaridade.

Na primeira categoria, a abordagem é dividir uma imagem com base em mudanças abruptas de intensidade, como bordas em uma imagem. As abordagens principais na segunda categoria são baseadas na partição de uma imagem, em regiões que são semelhantes de acordo com um conjunto de critérios predefinidos. O limiar, o crescimento da região e a divisão e fusão de regiões são exemplos de métodos nessa categoria [10].

2.2.7 Detecção de Descontinuidade

A maneira mais comum de procurar descontinuidade é executar uma máscara através da imagem da maneira descrita na seção 2.2.4. Para uma máscara $3x3$, como a vista na Figura 2.6, este procedimento envolve a computação da soma dos produtos dos coeficientes, com os níveis de cinza contidos na região abrangida pela máscara [10].

A resposta da máscara em qualquer ponto na imagem é dada por:

$$R = W_1 Z_1 + W_2 Z_2 + \ldots + W_9 Z_9 \tag{2.4}$$

onde Z_i é o nível de cinza do *pixel* associado com o coeficiente da máscara W_i. A resposta da máscara é definida em relação à sua localização central.

2.2.8 Detecção de Linha

Considerando as máscaras mostradas na equação 2.5, da esquerda para direita, caso a primeira máscara fosse movida em torno de uma imagem, ela responderia mais fortemente a linhas (um *pixel* de espessura) orientadas horizontalmente. Com um fundo constante, a resposta máxima teria resultado quando a linha passasse através da linha do meio da máscara. A segunda máscara responde melhor a linhas orientadas a $+45°$; a terceira máscara a linhas verticais; e a quarta máscara para linhas na direção $-45°$. Vale ressaltar que os coeficientes em cada máscara somam zero, indicando uma resposta zero das máscaras em áreas de nível de cinza constante [10].

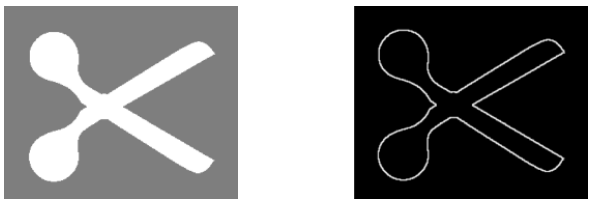
$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \tag{2.5}$$

2.2.9 Detecção de Borda

A detecção de bordas é a abordagem mais comum para detectar descontinuidades significativas no nível de cinza. Uma borda é um conjunto de *pixels* conectados que se encontram na fronteira entre duas regiões. As bordas são regiões da imagem onde ocorre uma mudança de intensidade em certo intervalo do espaço, em determinada direção. Isto corresponde a regiões de alta derivada espacial, que contêm alta frequência espacial [10].

Um operador que é sensível às variações abruptas de intensidade, operará como um detector de bordas. Um operador de derivada faz exatamente esta função. Uma interpretação de derivada seria a taxa de mudança de uma função, pois a proporção de variação dos níveis de cinza em uma imagem é maior perto das bordas e menor em áreas constantes. Ao obter os valores de intensidade da imagem e calcular os pontos onde a derivada é um ponto de máximo, tem-se a marcação das bordas [11].

Dado que as imagens são representadas em duas dimensões, é importante considerar mudanças nos níveis de cinza em várias direções. Por esta razão, derivadas parciais das imagens são usadas, com as respectivas direções X e Y. Uma estimativa da direção atual da borda pode ser obtida usando as derivadas X e Y como os componentes da direção ao longo dos eixos, computando-se a soma dos resultados



(a) Imagem de Entrada (b) Imagem de Saída

Figura 2.9: Detecção de borda.

parciais. O operador envolvido é o gradiente: se a imagem é vista como uma função de duas variáveis $f(x, y)$, então o gradiente é definido como:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \tag{2.6}$$

A magnitude do gradiente é dada por:

$$|\nabla f(x, y)|^2 = \left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \tag{2.7}$$

Conforme a equação 2.7, a utilização da função magnitude do gradiente na detecção de bordas é computacionalmente muito custosa. O que se faz na prática é a aproximação da equação acima para a forma:

$$\nabla f \approx |Gradiente_x| + |Gradiente_y| \tag{2.8}$$

Geralmente, a magnitude do gradiente tem um valor alto na borda e baixo no interior da região, o que possibilita a determinação do contorno de uma imagem.

Na Figura 2.9a é mostrada uma imagem fonte à qual será aplicada a operação de detecção de borda. Na Figura 2.9b é mostrado o resultado da operação de detecção de borda, neste caso, o gradiente da imagem, $|G_x| + |G_y|$.

O operador de Roberts, aproxima o gradiente de uma imagem por meio de diferenciação discreta [10]. A máscara de Roberts é uma matriz $2x2$. Para realizar a detecção de borda com a máscara de Roberts nas direções horizontal e vertical, a imagem inteira é convoluída com os operadores definidos na equação 2.9.

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{2.9}$$

A máscara de Prewitt é um operador de diferenciação discreta. Este operador usa duas máscaras $3x3$ para calcular os valores derivados aproximados nas direções horizontal e vertical. Na equação 2.10 podem ser visualizados os operadores de Prewitt.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.10}$$

A máscara de Sobel é similar a máscara de Prewitt. A diferença está relacionada ao peso dos valores de *pixel* em torno da região da borda, aumentando a intensidade da borda. Na equação 2.11 podem ser visualizados os operadores de Sobel.

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.11}$$

2.2.10 *Thresholding*

Suponha que o histograma de nível de cinza mostrado na Figura 2.10, corresponde a uma imagem, $f(x, y)$, composta de objetos claros sobre um fundo escuro, de modo que os *pixels* do objeto e do fundo têm níveis de cinza agrupados em dois modos dominantes. Vale ressaltar que os níveis de cinza variam entre o preto como a menor intensidade $f(x, y)$ e o branco como a maior intensidade. Uma maneira de extrair os objetos do fundo é selecionar um *threshold* (limiar) T que separa esses modos. Então qualquer ponto (x, y) para qual $f(x, y) > T$ é chamado de ponto do objeto; caso contrário, o ponto é chamado de ponto de fundo [10].

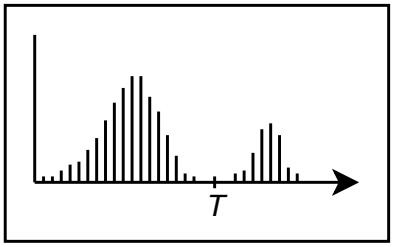


Figura 2.10: Histograma [10].

2.2.11 Resolução de Imagem

O número de *pixels* contidos numa imagem digital é expresso usando um termo chamado resolução. Este termo refere se ao número de *pixels* contidos na imagem, incorporando a largura e a altura. A resolução dá às imagens digitais suas dimensões 2D.

A resolução de imagem é expressa usando dois números (a largura ou x e a altura ou y) com um sinal de versus (x) no meio. Por exemplo, a resolução VGA comum seria expressa como $640x480$. Na tabela 2.1 pode-se visualizar alguns padrões de resoluções de vídeo.

Tabela 2.1: Padrões de resoluções de vídeo

Padrão de Vídeo	Resolução
VGA	$640x480$
SVGA	$800x600$
XGA	$1024x768$
SXGA	$1280x1024$
UXGA	$1600x1200$
QXGA	$2048x1536$
Full Resolution	$2592x1944$

2.2.12 Tipos de Imagens Digitais

Uma imagem pode ser classificada em 4 tipos: Binária, Escala de Cinza, RGB (*True Color*) e Indexada. Na imagem binária cada *pixel* é somente preto ou branco. Como existem apenas dois valores possíveis para cada *pixel*, têm-se apenas um bit por *pixel*.

Essas imagens podem, portanto, ser muito eficientes em termos de armazenamento. Uma imagem binária pode ser visualizada na Figura 2.11, na qual têm-se 2 cores, branco para as bordas e preto para o fundo.

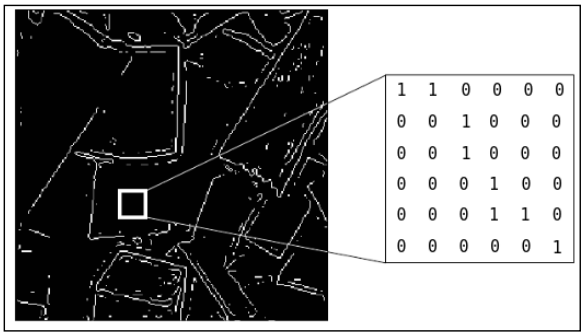


Figura 2.11: Imagem binária [15].

Na imagem em escala de cinza cada *pixel* é um tom de cinza, normalmente de 0 (preto) a 255 (branco). Esse intervalo significa que cada *pixel* pode ser representado por oito bits ou exatamente um byte. Esse é um intervalo muito natural para o manuseio do arquivo de imagem. Na Figura 2.12 é mostrada uma imagem em escala de cinza.

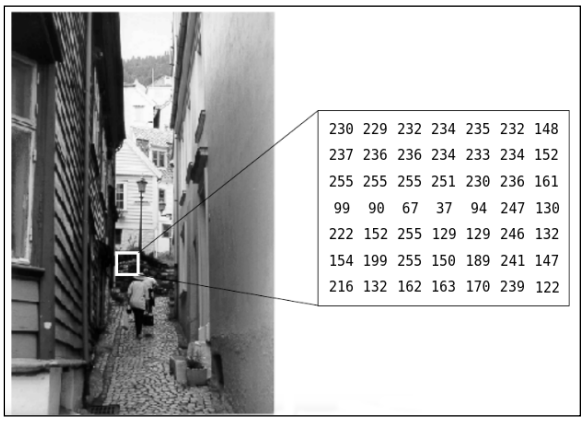


Figura 2.12: Imagem em escala de cinza [15].

Na imagem RGB cada *pixel* tem uma cor específica. Essa cor pode ser descrita por uma quantidade de vermelho, verde e azul. Se cada um desses componentes tiver um intervalo de 0 a 255, isso dará um total de $256^3 = 16.777.216$ cores possíveis diferentes na imagem. Como o número total de bits necessários para cada *pixel* é 24, essas imagens também são chamadas de imagens coloridas de 24 bits. Na figura 2.13 é mostrada uma imagem no formato RGB.

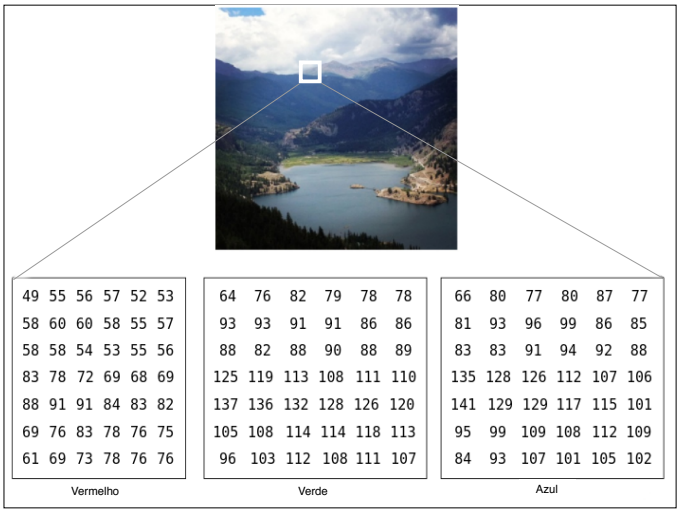


Figura 2.13: Imagem no formato RGB [15].

Uma imagem indexada tem uma matriz de mapa de cores associada. Os valores de *pixel* na imagem são índices diretos em um mapa de cores. Cada posição do mapa de cores armazena três valores em ponto flutuante no intervalo [0,1], especificando os componentes vermelho, verde e azul de uma única cor. Geralmente, as paletas possuem 2, 4, 16 ou 256 cores, representadas, respectivamente, por 1, 2, 4 ou 8 bits.

2.2.13 Tamanho dos Dados de uma Imagem

A quantidade de dados em uma imagem é relativamente grande, e isso afeta quase todos os aspectos do processamento de imagens e vídeos. O volume de dados é um grande problema no processamento, armazenamento, transmissão e exibição de informações de imagem e vídeo.

Considerando uma imagem binária com resolução de $512x512$, têm-se que o número de bits pode ser expresso por:

$$nbits = 512 * 512 * 1 \tag{2.12}$$

Dessa forma, pode-se afirmar que a imagem possui 262.144 bits, ou 32.768 bytes. Considerando uma imagem em escala de cinza com resolução $512x512$, têm-se que o número de bits pode ser expresso por:

$$nbits = 512 * 512 * 8 \tag{2.13}$$

Dessa forma, pode-se afirmar que a imagem possui 2.097.152 bits, ou 262.144 bytes. Considerando uma imagem no formato RGB (*True Color*) com resolução $512x512$, têm-se que o número de bits pode ser expresso por:

$$nbits = 512 * 512 * 24 \tag{2.14}$$

Dessa forma, pode-se afirmar que a imagem possui 6.291.456 bits, ou 786.432 bytes.

2.3 Vídeo

Um vídeo analógico pode ser representado como um sinal contínuo que varia ao longo do tempo, o qual pode ser expresso por $I(x, y, t)$, sendo x e y as coordenadas espaciais contínuas e t o tempo contínuo. Um vídeo digital pode ser expresso por $F(x, y, t)$, sendo x e y as coordenadas espaciais discretas do quadro de número t, também discreto. O conteúdo de um vídeo digital é uma sequência de imagens digitais chamadas de *frames* ou quadros, como pode ser visto na Figura 2.14. Assim como para a imagem digital, mostrado na seção 2.2.1, os *frames* são obtidos por amostragem e quantização do vídeo analógico [11].

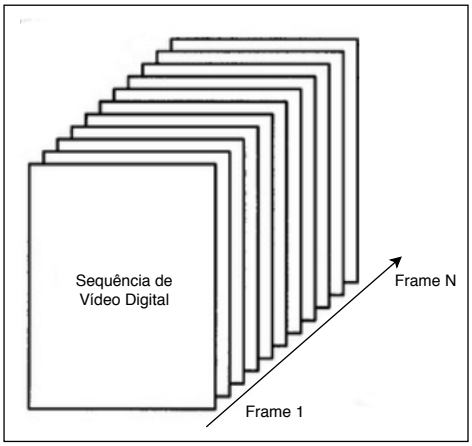


Figura 2.14: Sequência de vídeo digital [12].

Um vídeo digital pode ser obtido usando uma câmera de vídeo. A maioria das câmeras de vídeo digitais atuais usa sensores CCD. As câmeras digitais reproduzem a imagem ao longo do tempo, resultando em quadros discretos. Cada quadro consiste

em valores de saída de uma matriz CCD, que é por natureza discreta nas dimensões horizontal e vertical.

Um sinal de vídeo digital pode ser caracterizado por [11]:

- A taxa de *frames* ou quadros;
- O número de linhas; e
- O número de amostras por linha.

A taxa de *frames* define o intervalo de amostragem temporal ou intervalo de quadros. O número de linhas define o intervalo de amostragem vertical, ou seja, a altura da imagem. O número de amostras por linha define o intervalo de amostragem horizontal, ou seja, a largura da imagem.

2.3.1 *Scanning*

Scanning é um método usado pelos sistemas de vídeo, como parte do processo de conversão de imagens ópticas em sinais elétricos. Este método realiza uma varredura por toda imagem, iniciando no canto superior esquerdo da mesma, movendo-se na direção horizontal através do *frame*, formando uma linha de varredura. Em seguida, ocorre o retorno para a borda esquerda do *frame*, iniciando a varredura da próxima linha. Depois que a última linha for varrida, ou seja, quando o ponto de detecção atingir a parte inferior da imagem, o processo de varredura retorna ao canto superior esquerdo da imagem.

Este método pode ser realizado através de duas técnicas:

- *Progressive scanning*, na qual cada imagem é digitalizada em uma única passagem, chamada *frame*.
- *Interlaced scanning*, onde cada quadro é digitalizado em duas passagens verticais sucessivas, intercalando-se linhas pares com ímpares.

2.3.2 Sinais de Temporização do Fluxo de Vídeo

Fluxo de vídeo é uma série de imagens sucessivas. Cada quadro de vídeo consiste em *pixels* ativos e em branco. O sinal de sincronização horizontal *hsync* indica o início

da próxima linha. Começando do topo, todas as linhas ativas na área de exibição são escaneadas desta forma. Depois que todo o *frame* de vídeo ativo for escaneado, o sinal de sincronização vertical *vsync* indicará o início de um novo quadro de vídeo.

O intervalo de tempo ao final de cada linha, durante o qual o sinal de vídeo deve ser desativado, antes de uma nova linha ser escaneada, é conhecido como *Blanking Interval*. A taxa na qual as imagens são exibidas é conhecida como *frame rate*. A Figura 2.15 apresenta os sinas de temporização do fluxo de vídeo.

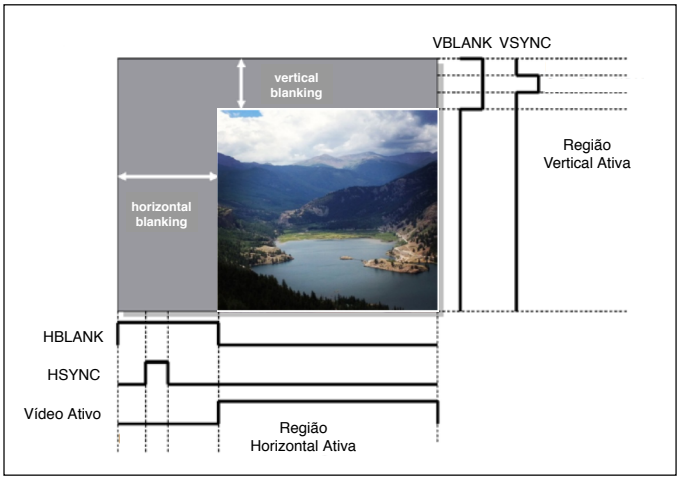


Figura 2.15: Sinais de temporização do fluxo de vídeo [12].

2.4 Processamento de Imagens em Tempo Real

Um Sistema de Tempo Real é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos [16]. A reação dos sistemas de tempo real aos eventos vindo do ambiente externo ocorre em tempos compatíveis com as exigências do ambiente e mensuráveis na mesma escala de tempo. A concepção do sistema de tempo real é diretamente relacionada com o ambiente no qual está relacionado e com o comportamento temporal do mesmo [16].

Na perspectiva de processamento de imagens, um sistema de geração de imagens em tempo real é aquele que captura imagens regularmente, analisa essas imagens para obter alguns dados e, em seguida, usa esses dados para controlar alguma atividade, respeitando uma restrição temporal. Esse sistema envolve o processamento de grandes quantidades de dados de imagem em tempo hábil, com o objetivo de extrair informações úteis. Imagens digitais e vídeo são essencialmente sinais multi-

dimensionais, apresentando um grande volume de dados, exigindo uma quantidade significativa de recursos de computação e memória para seu processamento. Um dos desafios em sistemas de processamento de imagens em tempo real é como lidar com a quantidade de processamento e cálculos.

Os sistemas em tempo real são categorizados em dois tipos [17]:

- O sistema de tempo real crítico (*hard real time*), é considerado falho, caso a saída não seja produzida dentro do intervalo de tempo predefinido.
- O sistema de tempo real não-crítico (*soft real time*), define que o sistema não é considerado falho, caso a saída não seja produzida dentro do intervalo de tempo predefinido, porém o desempenho se deteriora.

2.5 Métodos para a Detecção de Objetos

De acordo com [18], as principais abordagens utilizadas para a detecção de objetos são *Background Subtraction*, *Frame Differencing*, *Temporal Differencing* e *Optical Flow*, como pode ser visualizado na Figura 2.16.

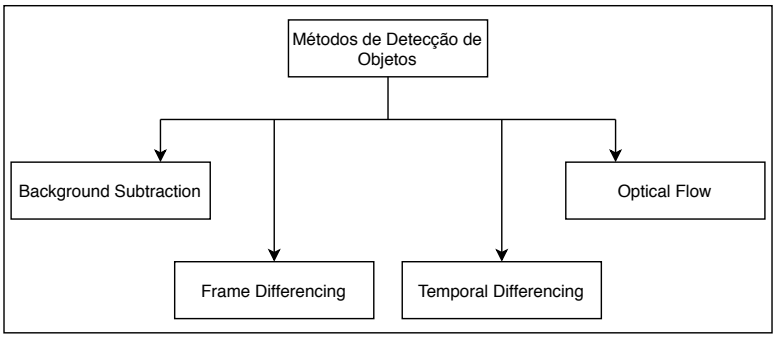


Figura 2.16: Métodos para a detecção de objetos.

Background Subtraction é o método mais usado para a detecção de objetos, o qual consiste no processo de separação de objetos em *foreground* (primeiro plano) e *background* (imagem de fundo) em uma sequência de quadros de vídeo. O conceito fundamental é a diferença entre o quadro atual e o quadro de referência. O quadro de referência também é chamado como imagem de fundo [18]. Embora esse método forneça um bom contorno dos objetos, ele é baseado em um plano de fundo estático, portanto, qualquer alteração na imagem será marcada como primeiro plano. Além

disso, o modelo de imagem de fundo deve ser atualizado ao longo do tempo, para se adaptar às mudanças dinâmicas da cena. Na Figura 2.17 pode ser visualizado o fluxograma do método *Background Subtraction*.

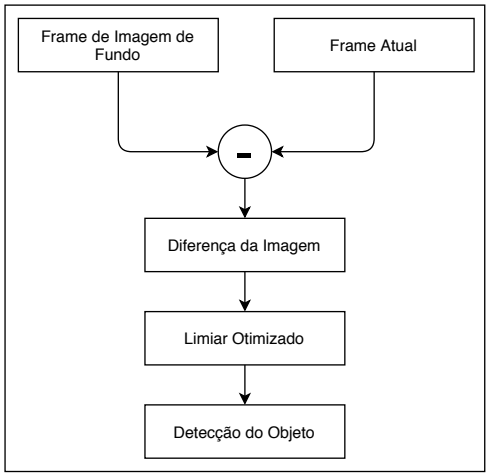


Figura 2.17: Conceito geral do método *Background Subtraction*.

Segundo [19], o *Frame Differencing* é o método que identifica a presença de um objeto considerando a diferença entre dois quadros, subtraindo a segunda imagem do primeiro quadro de imagem, usando o operador de subtração de imagem nos quadros consecutivos, para obter a saída desejada. O algoritmo pode ser subdividido em três partes. O passo inicial é a seleção de opções de imagem de fundo. O segundo passo é a operação de subtração aritmética e o terceiro passo é a seleção de limiar adequado. O método deve retornar o contorno do objeto identificado. Na Figura 2.18 pode ser visualizado o fluxograma do método *Frame Differencing*.

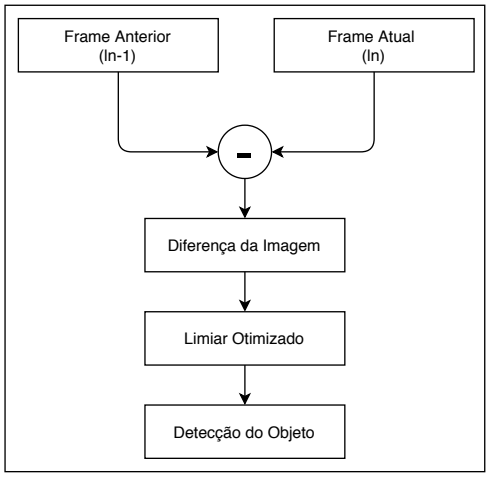


Figura 2.18: Conceito geral do método *frame differencing*.

De acordo com [20], a diferenciação temporal ou *Temporal Differencing* é baseada na diferença de quadros, que tenta detectar regiões em movimento usando a diferença de quadros consecutivos (dois ou três), em uma sequência de vídeo. Este método é adequado para o caso de objeto movimentando-se em um cenário de fundo estático. Portanto, a diferenciação temporal é boa em fornecer informações iniciais de áreas de movimento, no entanto as regiões convergentes podem ser detectadas como objetos em movimento, portanto, a saída pode conter ruídos devido a um erro de estimativa de movimento.

O método *Optical Flow* baseia-se no cálculo do campo de fluxo óptico do quadro de imagem ou vídeo. O agrupamento é realizado com base nas informações de distribuição de fluxo óptico obtidas da imagem. Para cada *pixel*, um vetor de velocidade é calculado, dependendo da direção do movimento do objeto e da rapidez com que o *pixel* se move pela imagem. Este método permite obter o conhecimento completo sobre o movimento do objeto, sendo útil para determinar o movimento do objeto a partir da imagem de fundo. A desvantagem deste método é a grande quantidade de cálculos que são necessários para obter informações sobre o fluxo óptico.

2.6 Trabalhos Correlatos

A detecção de objetos é um componente essencial para várias aplicações de visão computacional e processamento de imagem. Nesse contexto, alguns trabalhos correlatos já foram desenvolvidos, com a utilização de algoritmos encontrados na literatura. Nesta seção, serão abordados alguns destes trabalhos que apresentam conceitos e técnicas similares às utilizadas neste projeto.

No trabalho proposto em [21] foi desenvolvido um sistema que detecta e classifica objetos numa imagem. O projeto implementa o algoritmo SIFT (*Scale Invariant Feature Transform*) como uma aplicação executada no sistema operacional Ubuntu Linux, na placa de desenvolvimento Zedboard FPGA. Com a estrutura apresentada, foi possível realizar a detecção e classificação de objetos usando a biblioteca OpenCV. O sistema é composto por funções desenvolvidas tanto em software (algoritmo SIFT), quanto em hardware. O autor não informa os dados referentes ao desempenho, quando comparado ao processamento serial usando uma CPU, tampouco os dados

da resolução da imagem ou frequência máxima do circuito em hardware.

Segundo [22], foi desenvolvido um sistema que implementa o rastreamento de objetos, implementando o algoritmo de Canny para detecção de bordas, usando o FPGA Xilinx Spartan-III XC3S500E. O papel do algoritmo de rastreamento é a análise dos quadros de vídeo, para estimar o movimento dos parâmetros. Esses parâmetros caracterizam a localização do objeto. O sistema suporta o processamento de imagens com a resolução de $640x480$ *pixels*. A taxa de quadros de vídeo processados é de 1 quadro por segundo. O autor não cita os dados referentes ao desempenho, quando comparado ao processamento serial usando uma CPU ou a frequência máxima do circuito em hardware.

De acordo com [23], foi proposto e implementado um sistema de detecção de objetos no Xilinx MicroBlaze *soft-core processor*. O sistema implementa a técnica *background subtraction*, que permite que o objeto seja identificado através da diferença da imagem atual e o plano de fundo da imagem. Neste projeto foi utilizada a ferramenta XPS (*Xilinx Platform Studio*), para escrever o algoritmo usando a linguagem *C*. A ferramenta XPS é usada para a integração entre o software e o Xilinx MicroBlaze *soft-core processor*. O dispositivo FPGA utilizado neste trabalho foi o SPARTAN-3 XC3S200TQ-144. As imagens são obtidas no dispositivo FPGA através do protocolo RS-232. O sistema suporta o processamento de imagens com as resoluções de $320x240$ *pixels* e $160x120$ *pixels*. A taxa de quadros de vídeo capturados pode ser de 15 quadros por segundo ou 30 quadros por segundo. O autor não cita os dados referentes ao desempenho, quando comparado ao processamento serial usando uma CPU ou a frequência máxima do circuito em hardware.

No trabalho proposto em [24], foi desenvolvida uma arquitetura de hardware para a detecção de objetos, usando a linguagem de descrição de hardware VHDL e implementada usando o dispositivo FPGA Spartan-6 XC6SLX45-2csg324. O sistema implementa a técnica *background subtraction* modificada. Neste método, a segmentação de objetos é feita usando a subtração. Esta técnica calcula a diferença de duas imagens usando *pixel* por *pixel*, e continua por todos os *pixels* no quadro de vídeo. O sistema suporta o processamento de imagens com a resolução de $256x256$ *pixels*, podendo operar com uma frequência máxima de 164 MHz. O autor não cita os dados referentes ao desempenho, quando comparado ao processamento serial

usando uma CPU.

O diferencial deste trabalho em comparação aos trabalhos correlatos é a implementação do sistema de detecção de objetos em FPGA utilizando conceitos abordados nos métodos *Frame differencing*, *Temporal Differencing* e *Background subtraction*. O projeto de circuito digital foi codificado usando a linguagem Verilog, o ambiente de verificação foi desenvolvido e estruturado com a metodologia UVM e System Verilog e a prototipação realizada no kit FPGA Intel DE2-115. Com relação ao desempenho foram obtidos resultados com um *speedup* de 310x para a implementação do algoritmo de detecção de objetos em FPGA, quando comparada à implementação no software Matlab.

Capítulo 3

Metodologia do Projeto de Circuitos Integrados Digitais

O tradicional fluxo de projeto, para a implementação de um sistema completo em um dispositivo FPGA (*Field-Programmable Gate Array*), adota uma abordagem *top-down*, que apresenta uma divisão em níveis bem definidos. Segundo [25], atualmente os fornecedores de FPGA provêm um conjunto completo de ferramentas que permitem a automatização de tarefas, abrangendo desde as etapas de especificação do projeto até a geração do *bitstream*, para a programação do dispositivo FPGA.

O aumento na capacidade com relação à recursos e desempenho de dispositivos FPGA tem possibilitado o uso de tais dispositivos em projetos muito mais complexos, visando o desenvolvimento das aplicações utilizando o fluxo de projeto em FPGA. De acordo com [26], a utilização de arquiteturas de hardwares reconfiguráveis tem crescido entre os projetistas de diferentes áreas como robótica, telecomunicações, processamento digital de sinais e imagens, equipamentos médicos e automotiva, bem como na área de projetos de circuitos integrados (emulação/verificação). Segundo [9], foram obtidos avanços com a utilização de dispositivos FPGAs para a aceleração da computação de alto desempenho.

A facilidade para reconfigurar e recompilar um circuito digital, descrito em código RTL no dispositivo FPGA, reduz o tempo gasto para identificação e correção de erros, proporcionando mais tempo de verificação funcional no sistema. Essa característica pode acelerar o ciclo de verificação do projeto, bem como o ciclo de integração do sistema, ocasionando uma redução no tempo de desenvolvimento do

projeto. No entanto, essa melhora na produtividade somente será alcançada com a aplicação de uma metodologia de projeto adequada.

3.1 Arquitetura de Hardware Reconfigurável

Segundo [25], um FPGA é um dispositivo lógico que contém uma matriz bidimensional de células lógicas genéricas e programáveis. Uma célula lógica pode ser programada para executar uma função simples, bem como uma configuração pode ser personalizada, para prover interconexões entre as células lógicas. Um dispositivo FPGA pode fornecer diversos blocos de entrada/saída e suporte para vários padrões de entrada/saída. Estes circuitos fazem a interface entre os blocos internos da FPGA e os pinos externos.

FPGA usa o conceito de um bloco lógico complexo. A CLB (*Configurable Logic Blocks*), consiste num circuito para implementar a lógica e um CLB típico é mostrado na Figura 3.1. Cada bloco lógico é contornado por canais de roteamento, conectados por meio de matriz de chaveamento ou de interconexões. A matriz de chaveamento conecta os fios nos canais adjacentes, através de transistores de passagem ou *buffers* bidirecionais [27].

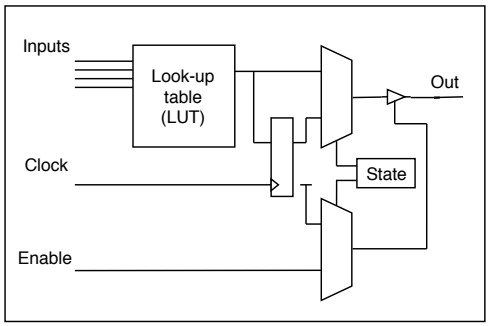


Figura 3.1: Bloco lógico configurável FPGA [28].

O CLB possui uma *look-up table* (LUT) que pode ser configurada para fornecer um tipo específico de função lógica, quando programado. Existe também um *flip-flop* do tipo D com *clock*, que permite que o CLB seja combinacional (sem *clock*) ou síncrono (com *clock*), além de um sinal de habilitação.

Um FPGA típico terá centenas ou milhares de CLB, de diferentes tipos, em um único dispositivo, permitindo que dispositivos muito complexos sejam implementa-

dos em um único *chip*, bem como configurados facilmente. A estrutura conceitual de um dispositivo FPGA pode ser vista na Figura 3.2.

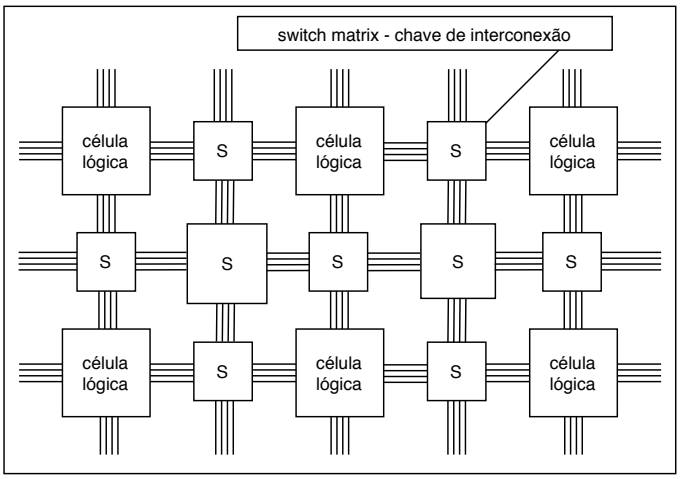


Figura 3.2: Estrutura conceitual de um dispositivo FPGA.
Fonte: Adaptado de [27].

Um projeto de circuito digital personalizado pode ser implementado especificando a função de cada célula lógica, possibilitando a conexão de cada chave programável. Uma vez que o projeto e a síntese são concluídos, pode-se usar um cabo adaptador simples, para baixar a configuração para o dispositivo FPGA e obter o circuito personalizado. Esta característica torna o dispositivo FPGA reprogramável.

3.2 Fluxo de Projeto em FPGA

Inicialmente é desenvolvido o documento de especificação, onde são definidas todas as funcionalidades do sistema, as interfaces internas e externas, os objetivos da aplicação e as restrições (*constraints*) que devem ser cumpridas para que o objetivo seja alcançado. Paralelamente à especificação do projeto, segundo [29], o sistema pode ser modelado e representado em uma ferramenta dedicada de alto nível como Matlab, ou linguagens de programação como C/C++. Esta modelagem é utilizada para a validação dos conceitos, e posteriormente poderá ser usada como Modelo de Referência, para a verificação funcional.

O próximo nível é o desenvolvimento da microarquitetura e nesta etapa são especificados os detalhes da arquitetura, estrutura dos blocos, funções e interfaces. Alguns recursos como diagramas de blocos, fluxogramas e gráficos, podem ser usados

para obter uma melhor representação da microarquitetura.

Baseado na microarquitetura desenvolvida, é iniciado o estágio de codificação RTL (*Register-Transfer Level*), através de uma HDL (*Hardware Description Language*). Esse processo, resulta numa forma de descrição da operação do circuito digital, na qual o comportamento do sistema é definido, em termos do fluxo de sinais entre registradores e das operações realizadas, mediante etapas de controle definidas através de pulsos de *clock*. Nesta etapa, os projetistas podem utilizar módulos proprietários codificados em RTL disponibilizados por diversas empresas com o objetivo de otimizar o desenvolvimento, chamados de IP (*Intellectual Property*). De acordo com [30], a detecção de uma função incorreta, tem se tornado um dos grandes desafios de todo o fluxo de projeto em FPGA.

A verificação funcional garante que o projeto de circuito digital execute as tarefas como pretendido pela arquitetura geral do sistema. A execução desta fase pode ser realizada de forma paralela ao estágio de codificação RTL. Inicialmente é elaborado um Plano de Verificação que inclui os objetivos, componentes e detalhes da estratégia de verificação. Baseado no plano de verificação, é desenvolvido um ambiente de verificação funcional e, através destes componentes, são criados mecanismos que visam a automatização de tarefas, referentes à ampla atividade de verificação. Atualmente é recomendado o uso de uma metodologia de verificação, como a metodologia UVM (*Universal Verification Methodology*) [31].

A próxima etapa é a síntese lógica, na qual é gerada uma *netlist* descrevendo todos os elementos combinacionais (portas lógicas simples e agregadas), sequenciais (*flip-flops*) e de interconexão (fios) que representam logicamente o circuito. As entradas desta etapa são o RTL e as restrições de desempenho (área, consumo e tempo). Segundo [27], esse processo pode ser otimizado, com o intuito de obter o menor consumo de energia, a menor área ou a maior velocidade. A etapa de síntese física abrange as seguintes fases: mapeamento, posicionamento e roteamento. A etapa de mapeamento determina como a lógica é mapeada para os diferentes tipos de componentes específicos disponíveis no FPGA. A etapa de posicionamento associa efetivamente componentes físicos do FPGA aos mapeamentos genéricos realizados na etapa anterior. A etapa de roteamento consiste no processo em que é feita a configuração das chaves comutadoras dos fios metálicos, que interconectam as células

lógicas, de armazenamento e de E/S do FPGA. No entanto, somente nos casos de projeto de circuitos digitais com alta complexidade, o projetista tem a necessidade de executar a etapa de síntese física de forma avançada, caso contrário a ferramenta é encarregada de realizar este procedimento, de forma automática e otimizada. O resultado deste estágio é um modelo do sistema em nível de portas, portanto o objeto gerado é novamente verificado funcionalmente, considerando o resultado da etapa de análise de tempo. Uma checagem sobre as restrições de tempo é realizada, a partir da especificação do arquivo SDC (*Synopsys Design Constraints*).

A última etapa é a geração do *bitstream*. Nesta fase são usadas ferramentas que executam tarefas automatizadas visando o mapeamento, alocação e roteamento do *design*, resultando no *bitstream* usado para programar a lógica e as interconexões do dispositivo FPGA. Após a configuração do FPGA, são feitos testes operacionais iniciais, bem como o monitoramento das funções implementadas. Na Figura 3.3 é mostrado o diagrama do fluxo de projeto.

3.2.1 Especificação do Sistema

Nesta etapa deve ser elaborado o documento de especificação do projeto. Uma especificação é um conjunto de requisitos que devem ser atendidos por um material, produto ou serviço. O documento de especificação é um balizador para todas as etapas seguintes do fluxo de projeto em FPGA. Alguns modelos [32] sugerem que o documento de especificação, contenha as seguintes informações: visão geral do projeto, objetivos, principais características, requisitos funcionais, requisitos não-funcionais, restrições, dentre outros tópicos relevantes.

Para a utilização do Fluxo de Projeto de Hardware Digital no sistema proposto neste trabalho, é necessário obter uma visão geral da aplicação proposta e determinar seus principais requisitos. O sistema de detecção de objetos em uma cena fixa em vídeos digitais, tem como propósito capturar imagens de vídeos, através da interface da câmera *TRDB-D5M Terasic CMOS sensor*, bem como processar e armazenar os quadros das imagens de vídeo e, por fim, apresentar os resultados através do monitor VGA (*Video Graphics Array*), utilizando o kit FPGA Intel DE2-115. A partir de um conjunto de quadros de imagens recebidos como entrada, o sistema poderá identificar os objetos realizando a comparação entre um quadro de imagem capturado, num

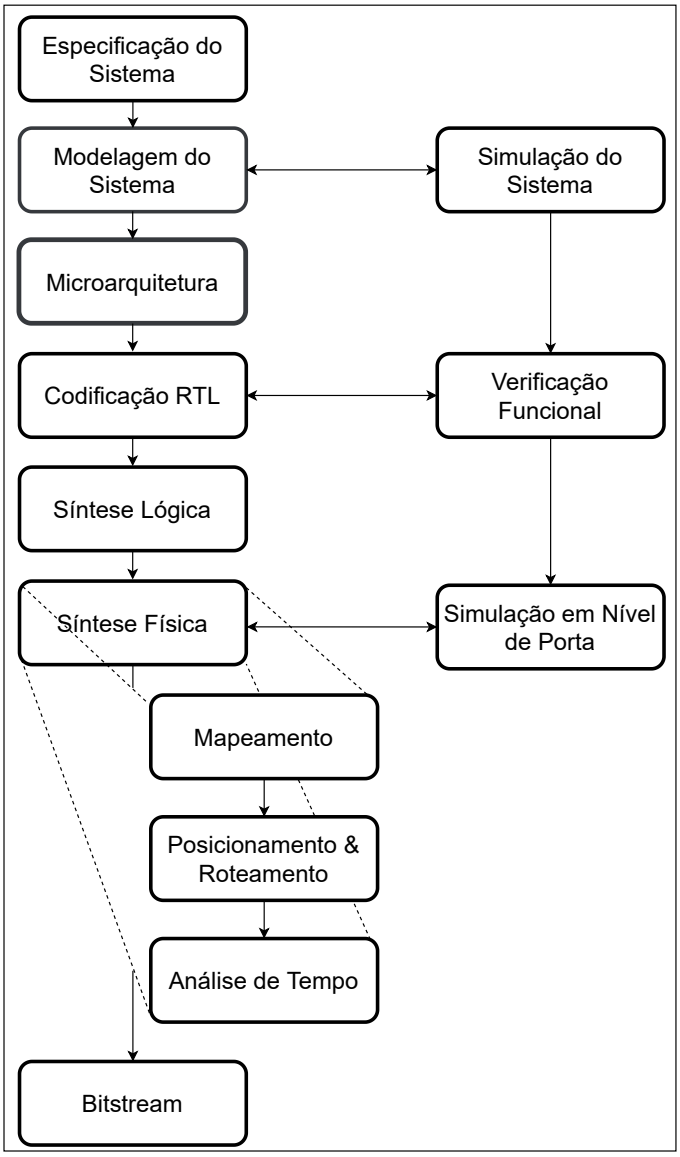


Figura 3.3: Fluxo de projeto de hardware digital.

período de tempo mais recente, com um quadro de imagem capturado, num período de tempo mais antigo. Na Figura 3.4, é apresentada uma visão geral do sistema proposto.

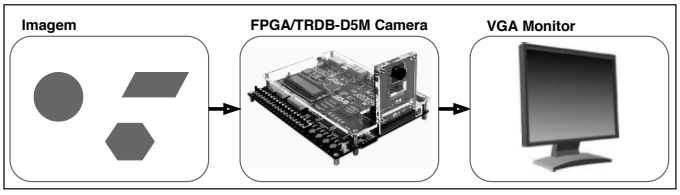


Figura 3.4: Visão geral do sistema proposto.

Os requisitos funcionais devem ser listados, para que o projetista prossiga com o

desenvolvimento do sistema. Para esta aplicação, alguns requisitos funcionais mais relevantes foram selecionados:

1. O processo de identificação de objetos é aplicado somente a um único quadro de imagem de vídeo capturado. A cada período de tempo determinado, um novo quadro de imagem de vídeo é capturado.

2. A identificação dos objetos é realizada através do processo de detecção de borda, utilizando o operador de Sobel.

3. O resultado obtido com a identificação do objeto é apresentado no monitor VGA, sendo o quadro da imagem de vídeo em RGB e o objeto visualmente destacado.

4. Após a identificação do objeto e a exibição no monitor VGA, o sistema pode reiniciar o processo através do *RESET*.

5. O conjunto de entradas de quadros de imagens de vídeo são armazenados na memória, para que seja possível a comparação entre os *pixels* de dois quadros de imagens distintos.

6. A resolução dos quadros da imagem de vídeo de entrada é limitada pelas configurações disponíveis da câmera TRDB-D5M. As resoluções disponíveis são $800x600$ e $640x480$.

7. O sistema implementa um módulo, capaz de converter uma imagem no formato RGB, para escala de cinza.

8. Os módulos que implementam o processamento de imagens se comunicam com os módulos IP (*Intellectual Property*) utilizados no projeto. Os módulos IP são CCD-CAPTURE, RAW2RGB, SDRAM-CTRL e VGA-CTRL [32].

9. O sistema utiliza o botão (*push button*) associado a KEY[0] do kit FPGA Intel DE2-115, para reiniciar todo o processo.

10. O sistema utiliza o botão associado a KEY[3] do kit FPGA Intel DE2-115, para inicializar todo o processo.

11. A memória SDRAM opera com uma frequência de *clock* de 100MHz, o módulo VGA-CTRL pode operar com uma frequência de *clock* de 40MHz para a resolução $800x600$, ou de 25MHz para a resolução $640x480$, como especificado em [33]. Todos os módulos restantes operam com uma frequência de 50MHz.

Alguns requisitos não-funcionais também foram selecionados:

1. O protótipo FPGA deve ser executado na plataforma Intel DE2-115.
2. O projeto deve ser descrito na linguagem Verilog-HDL.

3.2.2 Modelagem do Algoritmo de Processamento de Imagem

O comportamento do sistema pode ser validado utilizando ferramentas de modelagem e simulação, como o Matlab/Simulink, System Vue, dentre outras, bem como linguagens de programação como C/C++, usando recursos como SystemC, para descrever as principais funcionalidades do sistema.

A aplicação do Fluxo de Projeto de Hardware Digital, com relação à modelagem do sistema proposto neste trabalho, utilizou-se da linguagem dedicada Matlab para descrever o comportamento do sistema, desde o processo de aquisição da imagem, conversão da imagem no formato RGB para escala de cinza, detecção de bordas utilizando o operador de Sobel, e comparação entre os *pixels* de quadros de imagens distintos (tanto para a identificação dos objetos, quanto para a exibição do resultado, com o objeto destacado na imagem).

As funções referentes aos algoritmos de detecção de bordas e comparação entre os *pixels* foram implementadas em linguagem C, convertendo a representação numérica para a equivalente ao do código RTL correspondente. Tais implementações têm como objetivo proporcionar um Modelo de Referência para verificar a corretude funcional do código RTL. A Figura 3.5, apresenta o fluxo aplicado para a modelagem do algoritmo de processamento de imagens.

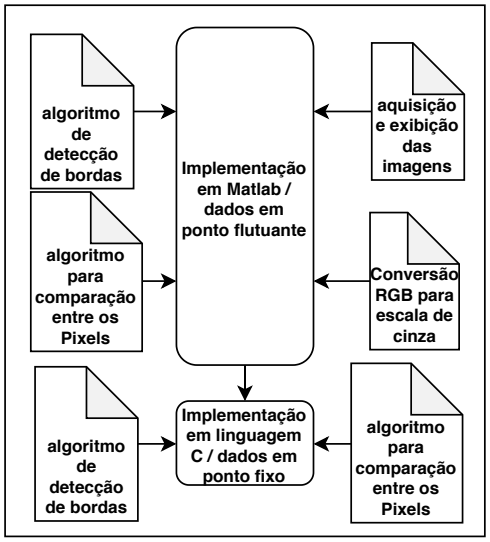


Figura 3.5: Fluxo da modelagem do algoritmo de processamento de imagens.

3.2.3 Microarquitetura

A microarquitetura apresenta os detalhes de todos os módulos implementados do sistema: as interfaces internas e externas dos blocos, os protocolos de comunicação utilizados, as informações de configuração sobre os módulos IP utilizados, a representação das máquinas de estados (FSM) dos blocos, entre outros. É importante apresentar as informações detalhadas sobre todos os sinais de entrada e saída de um determinado bloco, por exemplo, o nome do pino/porta, o tamanho em bits, a direção e a descrição da funcionalidade do pino/porta. Os espaços de endereçamento das memórias utilizadas devem ser definidos, bem como a arquitetura de interface entre as memórias e os módulos implementados.

A aplicação do Fluxo de Projeto de Hardware Digital, com relação a Microarquitetura desenvolvida, será apresentada com mais detalhes no capítulo 4.

3.2.4 Codificação RTL

Após o desenvolvimento da microarquitetura, o projeto de circuito digital deve ser codificado em alguma forma legível por humanos. A codificação RTL representa o projeto de circuito digital, usando uma Linguagem de Descrição de Hardware (HDL), e captura os aspectos essenciais do projeto que permitem a realização da simulação e síntese lógica em um FPGA. As HDL descrevem o hardware necessário para implementar o projeto de circuito digital. Para uma implementação em FPGA

de um algoritmo, é necessário descrever não apenas o algoritmo, mas também o hardware usado para implementá-lo. A maioria dos ambientes de desenvolvimento permite que a descrição de hardware seja elaborada, compilada e simulada no nível lógico, para verificar se ela se comporta da maneira pretendida. Para desenvolver e codificar um hardware digital, não basta apenas conhecer uma linguagem HDL, é necessário compreender também alguns conceitos sobre circuitos lógicos digitais.

3.2.4.1 Linguagens de Descrição de Hardware

As linguagens de descrição de hardware evoluíram na década de 1980, para abordar algumas das limitações das representações esquemáticas e para fornecer uma representação padrão do comportamento de um circuito [34]. Como o hardware é inerentemente paralelo, é necessário que uma HDL seja capaz de especificar e modelar o comportamento simultâneo, até o nível de porta lógica. Isso torna as HDL bastante diferentes das linguagens convencionais de programação, que são basicamente sequenciais.

As duas principais HDL são a linguagem VHDL (*Very High Speed Integrated Circuit HDL*) e a linguagem Verilog, ambas padrões IEEE. Elas são linguagens estruturais hierárquicas, na medida em que descrevem estruturalmente o hardware, em termos de blocos lógicos e suas interconexões. Elas também permitem uma descrição comportamental da funcionalidade do circuito, em vez de necessariamente identificar especificamente os circuitos lógicos. Uma descrição comportamental, como o próprio nome indica, descreve como o circuito deve se comportar ou responder a mudanças em suas entradas. É, portanto, o papel das ferramentas de síntese determinar a lógica real necessária para implementar o comportamento especificado [34].

Para a aplicação do Fluxo de Projeto de Hardware Digital, neste trabalho foi utilizada a linguagem Verilog. O Verilog contém um conjunto de primitivas integradas, que inclui desde portas lógicas, primitivas definidas pelo usuário, comutadores e lógica com fio. A linguagem também provê lógica de atrasos dos sinais de entrada/saída do dispositivo e checagem de temporização, para propósitos de simulação. [35].

A mistura de níveis de abstração é essencialmente fornecida pela semântica de dois tipos de dados [35]: *nets* e *variables*. Os *assignments* contínuos são usados para

direcionar um valor para uma *net* na modelagem de fluxo de dados. Os *assignments* procedurais ocorrem dentro de procedimentos como *always*, *initial*, *task* e *functions*, sendo usadas para colocar valores em variáveis. Um projeto consiste em um conjunto de módulos, cada um com uma interface de Entrada / Saída, e uma descrição de sua função, que pode ser estrutural, comportamental ou uma combinação de ambos. Esses módulos são formados em uma hierarquia e são interconectados com *nets*.

3.2.4.2 Circuitos Lógicos Digitais

A implementação bem-sucedida de algoritmos em um FPGA requer uma combinação de habilidades de software (algorítmicas) e de hardware (design de circuitos lógicos).

Circuitos lógicos digitais podem ser classificados em dois tipos [36]: combinacionais e sequenciais. Um circuito combinacional é projetado usando portas lógicas, nas quais a aplicação de entradas gera as saídas a qualquer momento. Um exemplo de um circuito combinacional é um somador, que produz o resultado da adição, como resultado da aplicação dos dois números a serem adicionados como entradas.

Um circuito sequencial, por outro lado, é projetado usando portas lógicas e elementos de memória conhecidos como *flip-flops*. O *flip-flop* é uma memória de um bit. Um circuito sequencial gera as saídas do circuito, com base nas entradas atuais e nas saídas (estados) dos elementos da memória. O circuito sequencial é basicamente um circuito combinacional com memória.

Comumente numa aplicação que envolve a implementação em FPGA, a estrutura básica de circuitos lógicos é a lógica intercalada com registradores (*flip-flops*), como pode ser visualizado na Figura 3.6. A estrutura apresenta uma série de blocos alternados de lógica e registradores. A estrutura é sequencial, como está implícito na Figura 3.6. A partir de uma perspectiva computacional, o objetivo principal de cada bloco de lógica é determinar o próximo valor que será armazenado nos registradores e a saída.

Cada bloco de lógica, pode receber uma entrada a partir de outros registradores, pinos de Entrada/Saída ou blocos de memória interna. As entradas, saídas e acessos à memória podem ser diretos ou via registradores. Os blocos de lógica combinacional não são sincronizados por um *clock*; toda sincronização por um *clock* é através dos registradores (e também da memória, se for síncrona). Portanto, os sinais devem

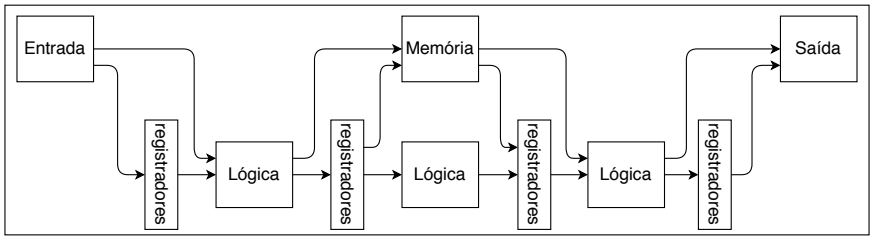


Figura 3.6: Estrutura de um circuito lógico digital.

se propagar através da lógica, antes que possam ser sincronizados nos registradores. Isso limita a velocidade do *clock* do projeto ao atraso de propagação através do bloco mais complexo.

Implícita na Figura 3.6 está a lógica de controle necessária para sequenciar as etapas dentro do circuito digital. Dependendo do algoritmo e sua implementação em circuito digital, nem todos os registradores devem ser atualizados a cada ciclo de *clock*. Isso pode ser obtido usando um *clock enable*, para ativar/desativar a atualização de registradores. Além disso, o cálculo para produzir um valor para um registrador pode não ser necessariamente o mesmo em todos os ciclos.

Uma implementação comum da lógica de controle é como uma máquina de estados finita (a máquina é finita, pois tem uma quantidade finita de estados), com o contexto codificado nos estados. O funcionamento de uma máquina de estados e os tipos serão explicados na próxima seção.

3.2.4.3 Máquina de Estados Finita

Máquinas de estados finita (FSM) são circuitos sequenciais usados em muitos sistemas digitais, para controlar o comportamento de sistemas e caminhos de fluxo de dados. Exemplos de FSMs incluem unidades de controle e sequência. Os tipos mais comuns de FSM são conhecidos como máquinas de estados de *Mealy* e *Moore*.

Uma máquina de estados finita tem um conjunto de estados e duas funções, chamadas de função de próximo estado e de saída [37]. O conjunto de estados corresponde a todos os valores possíveis do armazenamento interno. Assim, se houver n bits de armazenamento, existem 2^n estados. A função de próximo estado é uma função combinacional que, dadas as entradas e o estado atual, determina o próximo estado do sistema. A função de saída produz um conjunto de saídas a partir do estado atual e das entradas ou apenas do estado atual. Na Figura 3.7 pode ser

visualizada a estrutura básica de uma máquina de estados. As máquinas de estado são síncronas. Isso significa que o estado muda junto com o ciclo de *clock*, e um novo estado é calculado uma vez a cada *clock*. Assim, os elementos de estado são atualizados apenas na borda do *clock*.

Quando uma máquina de estados finita é usada como um controlador, a função de saída é frequentemente restrita a depender apenas do estado atual. Essa máquina de estado finita é chamada de máquina *Moore*. Se a função de saída pode depender do estado atual e da entrada atual, a máquina será chamada de máquina *Mealy*.

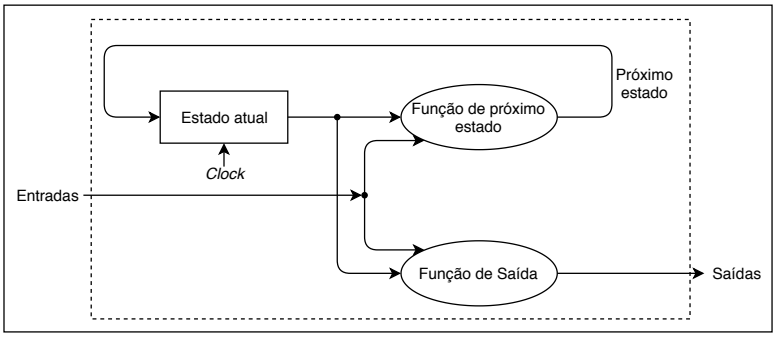


Figura 3.7: Estrutura básica de uma máquina de estados finita [37].

3.2.5 Verificação Funcional

A Verificação funcional é uma etapa fundamental nos fluxos de projeto em FPGA, pois assegura o correto funcionamento dos módulos implementados em RTL. Além disso, o processo de verificação visa encontrar erros de implementação e certifica que o projeto implementa a especificação. Inicialmente um plano de verificação é desenvolvido, sendo que as informações contidas são baseadas no documento de especificação do sistema. O plano de verificação contém uma lista de funcionalidades e testes, que devem ser realizados para validar as especificações do sistema. Os principais componentes de um plano de teste incluem informações como o escopo do plano de verificação, metodologia de verificação, descrição dos ambientes de verificação funcional e os critérios utilizados para avaliar a conclusão da verificação.

Para o processo de verificação é recomendado a utilização de uma metodologia de verificação, como a UVM (*Universal Verification Methodology*), que oferece uma estruturação e aplica as melhores práticas para verificação de forma eficiente e exaustiva. Uma das principais características do UVM é o fornecimento de componentes

de verificação reusáveis (UVC). O ambiente de verificação provê uma estrutura que implementa a verificação direcionada a cobertura (CDV), a qual combina a geração de testes automaticamente, estrutura de auto-verificação e métricas de cobertura.

3.2.5.1 Metodologia de Verificação Universal

A *Universal Verification Methodology* (UVM) é uma metodologia completa, que codifica as melhores práticas para verificação eficiente e exaustiva. Um dos princípios fundamentais da UVM é desenvolver e fornecer componentes de verificação reusáveis, também chamados de Componentes de Verificação UVM (UVC). A UVM é direcionada para verificar tanto projetos pequenos quanto projetos baseados em grandes sistemas baseados em IP *System-on-Chip* (SoC).

O código UVM é baseado na biblioteca *Open Verification Methodology* (OVM), com modificações feitas no topo do código OVM. A UVM é um padrão Accellera (e direcionado para o IEEE), fornecido em um formato de código aberto e testado para funcionar em todos os principais simuladores comerciais.

3.2.5.2 Estratégia de Verificação

A estratégia de verificação determina as estruturas e recursos que estarão dentro do *testbench*, bem como o objetivo da verificação. Um *testbench* consiste em uma hierarquia de módulos contendo componentes de verificação que estão conectados ao *Design Under Verification* (DUV). A meta da estratégia de verificação é garantir o seguinte:

1. Criar uma infraestrutura de *testbench* reusável e fácil de usar.
2. Criar uma arquitetura para o ambiente de verificação, capaz de estimular a maioria dos recursos do DUV.
3. Reduzir o tempo e o esforço gastos no desenvolvimento do ambiente.
4. Tornar o *testbench* compreensível pela equipe de projeto.
5. Evitar o profundo conhecimento do projeto de circuito integrado / FPGA do sistema pela equipe de verificação.

Do ponto de vista técnico, os objetivos incluem:

1. Definir os modelos de transação (objeto de estímulo, que serão convertidos em sinais DUV): itens de dados, sequências e hierarquia de sequências.

2. Selecionar as interfaces nas quais as transações serão enviadas e os pontos de interesse a serem monitorados.

3. Identificar os componentes ativos (*drivers*), componentes passivos (*monitors*, *scoreboards*) e a integração dos UVC.

4. Definir a arquitetura do ambiente, como o (*env*) conecta e controla os componentes citados anteriormente.

3.2.5.3 Componentes do Ambiente de Verificação

O ambiente de verificação funcional considera os seguintes componentes:

1. O Gerador de Estímulos, que é responsável por definir a estrutura do item de dados e casos testes.

2. O *Data Item*, que representa a entrada de dados do DUV (*Design Under Test*), o qual consiste no módulo codificado em RTL a ser verificado.

3. O *Driver*, entidade ativa que emula a lógica que insere os dados para o DUV.

4. O *Monitor*, entidade passiva que realiza uma amostragem dos sinais de entrada e saída do DUV. O *Monitor* coleta informações para cobertura funcional, e pode realizar a verificação de protocolos.

5. O *env*, que é um componente de contêiner para agrupar subcomponentes orientados em torno de um bloco ou em torno de uma coleção de blocos em níveis mais elevados de integração.

6. O *Scoreboard*, que é um componente fundamental do ambiente de verificação. Este componente coleta as transações enviadas pelo *Monitor* e realiza análises específicas sobre o dado coletado. O *Scoreboard* determina se o módulo verificado está funcionando corretamente. Este componente é responsável por implementar ou oferecer uma comunicação com o Modelo de Referência. Um Modelo de Referência, também referido como *Golden Reference Model*, recebe

o mesmo fluxo de estímulos que o DUV e produz fluxos de transação de boa resposta conhecidos. Por fim o *Scoreboard* avalia os resultados obtidos na saída do DUV com os resultados obtidos no Modelo de Referência.

Na Figura 3.8 é possível observar os componentes de um ambiente de verificação.

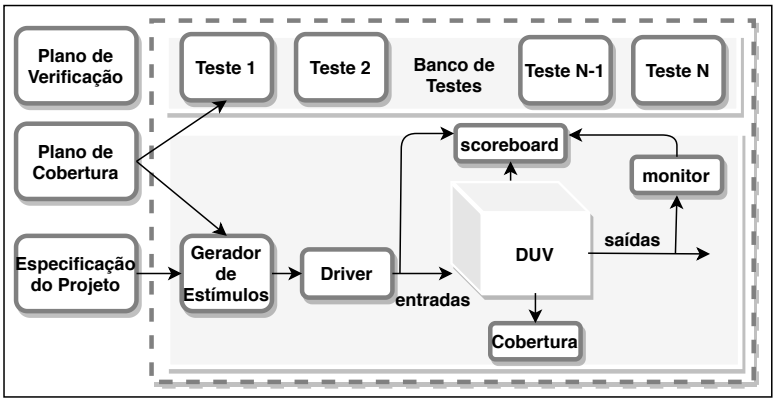


Figura 3.8: Estrutura do ambiente de verificação.

3.2.5.4 Verificação baseada em *Assertion*

Uma *Assertion* é uma confirmação positiva sobre uma propriedade do código RTL que se for considerada falsa, indica um erro. *Assertions* podem ser utilizadas para expressar o comportamento pretendido nas especificações, que podem ser interpretadas e analisadas por ferramentas. Como a propriedade apenas declara o comportamento, a *assertion* geralmente é usada para garantir que a implementação do código RTL tenha um comportamento correspondente à asserção.

A linguagem SystemVerilog fornece recursos de *assertions*, que são declarativos. Ou seja, afirmações não descrevem como verificar, mas apenas o que verificar. *Assertions* promovem metodologias sistemáticas, usando várias maneiras flexíveis de inserir verificações de projeto.

3.2.6 Síntese Lógica, Síntese Física, Simulação em Nível de Portas e Geração do *Bitstream*

Ferramentas de fornecedores de FPGA são usadas para realizar as atividades deste conjunto de etapas, mapeando, posicionando e roteando a *netlist* para os recursos do FPGA. A saída desse processo indicará se há recursos suficientes disponíveis no

FPGA de destino. Tal processo também fornece informações de tempo, com base nos recursos reais usados. A realimentação desse processo é usada para modificar os algoritmos ou a implementação, quando necessário, para atender às restrições de tempo.

As etapas são automatizadas e executadas por ferramentas como Intel Quartus Prime, Xilinx Vivado, entre outras. O projetista deve configurar a família do dispositivo e neste trabalho foi utilizada a Cyclone IV E. Inicialmente deve ser configurado um PLL, que usa o oscilador do dispositivo para criar uma frequência de *clock* constante, bem como deve ser realizada a atribuição dos pinos (*Pin Planner*), entre os sinais de entrada/saída do módulo codificado em RTL e os pinos do dispositivo FPGA. As entradas para a ferramenta são todos os arquivos codificados em RTL e o arquivo SDC (*Synopsys Design Contraints*), que a ferramenta utiliza para analisar as restrições de tempo.

3.2.6.1 Síntese Lógica

A Síntese lógica é iniciada com a elaboração do RTL, que identifica e/ou infere operações de caminho de dados, como adição, multiplicação, registradores, blocos de memórias e lógica de controle que é elaborada dentro de um conjunto de máquina de estados e/ou redes booleanas. Em seguida, têm-se a otimização independente de arquitetura, mapeamento de tecnologia e otimização específica de arquitetura.

3.2.6.2 Síntese Física

A etapa de posicionamento determina a localização de cada elemento na *netlist* mapeada. A etapa de roteamento realiza a conexão de todos os caminhos e sinais usando as interconexões disponíveis no dispositivo.

3.2.6.3 Simulação em Nível de Portas

A etapa de simulação em nível de portas permite analisar o circuito gerado, considerando informações mais realistas, tais como atrasos associados a cada um dos elementos lógicos, ou induzidos pela etapa de roteamento.

3.2.6.4 Geração do *Bitstream*

A última etapa é a geração do arquivo que contém as informações de programação para um FPGA. A ferramenta obtém o projeto mapeado, posicionado e roteado como entrada e gera o *Bitstream* para programar o FPGA.

Capítulo 4

Descrição do Sistema de Detecção de Objetos em uma Cena Fixa em Vídeos Digitais

4.1 Processo de Aquisição de Imagens

Os dados dos quadros de vídeo são capturados pela câmera TRDB-D5M Terasic Sensor CMOS, mostrada na Figura 4.1, e são encaminhados para o módulo de captura de imagem, onde as informações dos *pixels* válidas são extraídas.

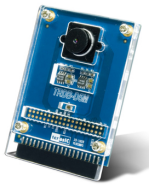


Figura 4.1: Câmera TRDB-D5M terasic sensor CMOS.

Sensores de imagem modernos geralmente são de dois tipos, *Complementary Metal-Oxide Semiconductors* (CMOS) ou *Charge-Coupled Devices* (CCD) [38]. No sensor do tipo CMOS, a matriz é tipicamente construída num circuito de *pixel* ativo contendo um fotodetector e amplificador que produz um sinal analógico para cada *pixel* enquanto é exposto. Durante a leitura, os sinais analógicos de cada linha da matriz são selecionados, amostrados e integrados, um por um, antes de serem convertidos pelo processador de sinal analógico. A Figura 4.2 mostra a matriz e

circuito utilizados numa câmera do tipo CMOS.

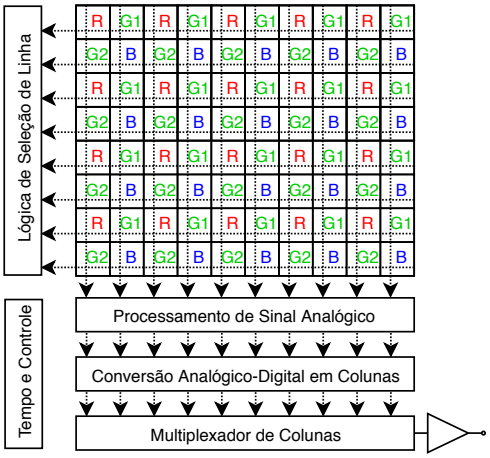


Figura 4.2: Estrutura de matriz de *pixels* e o *bayer pattern*.

Como pode ser visto na Figura 4.2, os valores da composição dos *pixels* são gerados de acordo com o padrão Bayer, *Green 1*, *Green 2*, *Red* e *Blue* (G1, G2, R, B). Estes dados são obtidos através da interface de comunicação, que pode ser visualizada na Figura 4.3. Esta interface contém informações sobre o *clock* e sinais de sincronização. Cada *pixel* obtido pelo sensor tem a resolução de 12 bits.

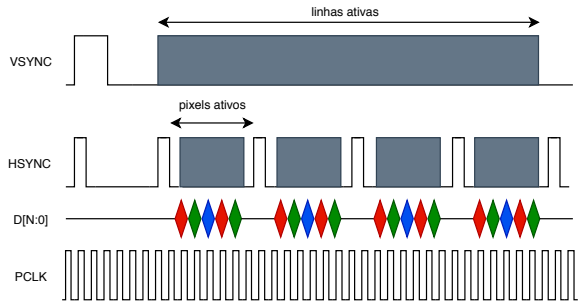


Figura 4.3: Interface padrão do sensor CMOS.

Os *pixels* do quadro de vídeo são lidos no modo *progressive scan*. Os *pixels* válidos na imagem são cercados pelas regiões de apagamento vertical e horizontal. Na Figura 4.3, o sinal *HSYNC* define os limites entre as linhas da região ativa, ou seja, o sinal está ativo em *HIGH* na região válida da imagem. O sinal *VSYNC* define a região ativa do quadro de vídeo. O sinal de *clock PCLK* determina a taxa de amostragem dos dados de *pixel*, para cada ciclo um novo valor de 12 bits é gerado.

Como pode ser observado na Figura 4.4, o hardware obtém um fluxo de dados, através da captura da imagem ótica pelo sensor de imagem. É importante ressaltar

que os dados devem ser primeiramente capturados e armazenados em seguida na memória. O processamento dos dados é feito posteriormente, através do acesso à memória SDRAM.

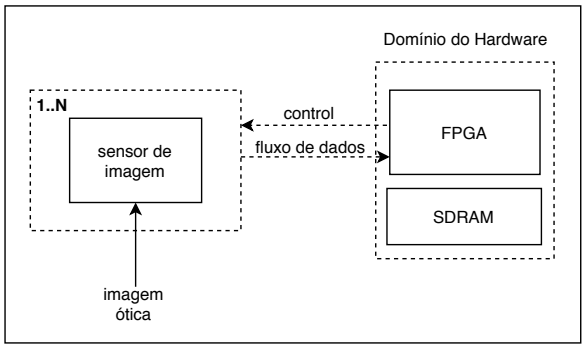


Figura 4.4: Processo de aquisição de imagens.

4.2 Descrição do Algoritmo de Detecção de Objetos em uma Cena Fixa

A aplicação empregada neste trabalho tem como objetivo implementar um algoritmo que identifique objetos, a partir de imagens capturadas por uma câmera. Na Figura 4.5 é apresentado o fluxograma do sistema, abordando apenas as principais funções, possibilitando uma visão geral do funcionamento. As funções **Processamento da Imagem** e **Geração/Atualização da Imagem** com o objeto detectado, apresentadas no fluxograma, exigem um grande esforço computacional, porque incluem o processamento de uma grande quantidade de *pixels*, utilizando operações lógicas e aritméticas. Tais funções serão explicadas com mais detalhes nas seções 4.3 e 4.4. Nesta seção, será explicado com detalhes o funcionamento de todo o sistema.

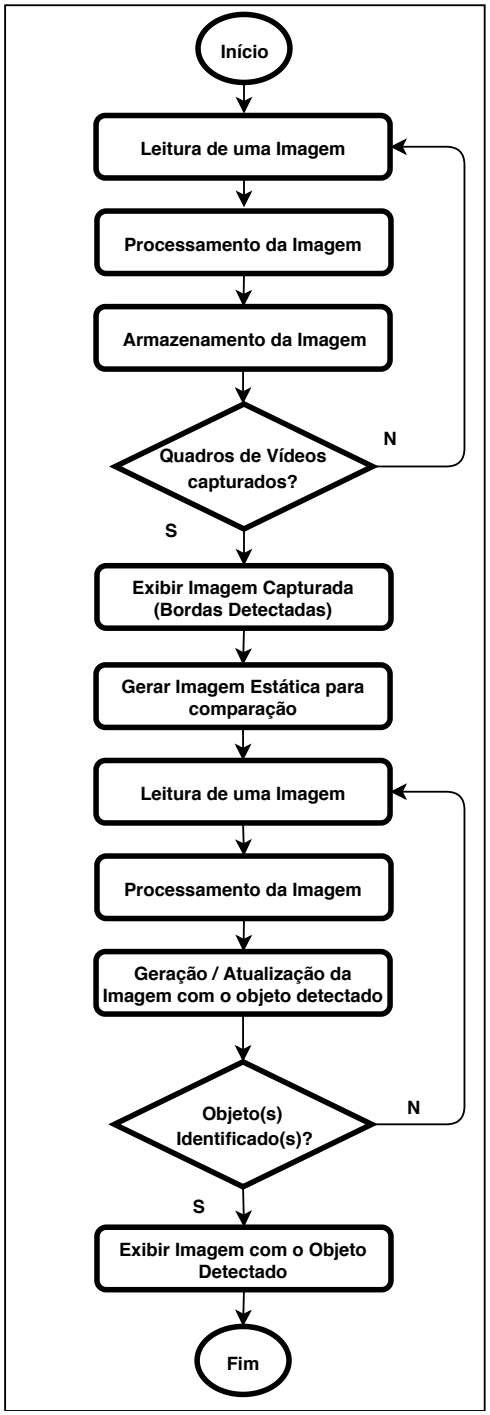


Figura 4.5: Processo de detecção de objetos em uma cena fixa.

Inicialmente, um quadro de vídeo **A.1** é capturado pela câmera e os dados da imagem são armazenados na memória. O quadro de vídeo **A.1** é convertido do formato RGB (Vermelho, Verde, Azul), para escala de cinza, a detecção de borda é aplicada, as bordas (na cor preta) são armazenadas na memória e o quadro de vídeo **A.1** é exibido no monitor. A 1ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.6.

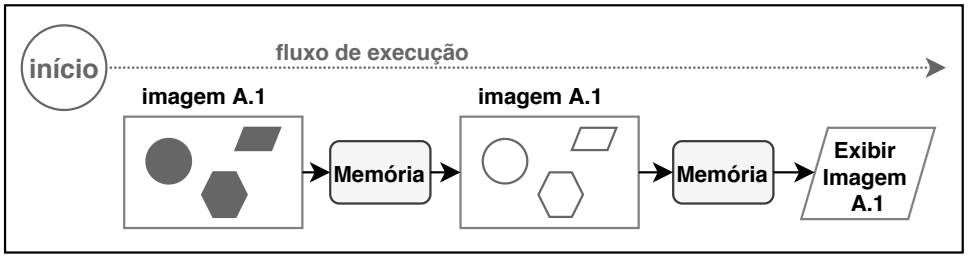


Figura 4.6: 1ª etapa do fluxo do algoritmo de detecção de objetos.

O sistema deve esperar um tempo, por exemplo, 1 segundo, para realizar uma nova captura do quadro de vídeo. Após o tempo de espera, um novo quadro de vídeo **B.1** é capturado pela câmera e os dados da imagem são armazenados na memória. O quadro de vídeo **B.1** é convertido do formato RGB, para escala de cinza, em seguida, a detecção de borda é aplicada e as bordas (na cor preta) são armazenadas na memória. A 2ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.7.

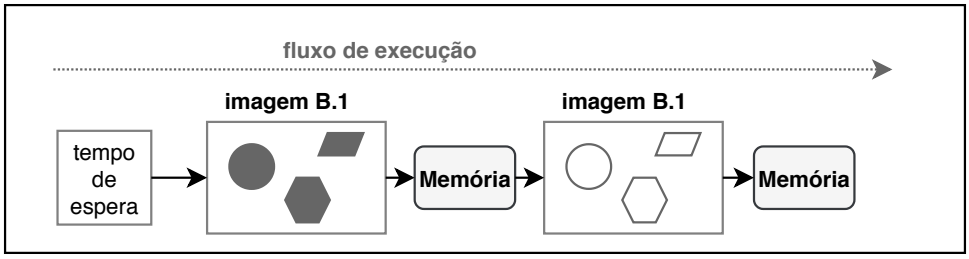


Figura 4.7: 2ª etapa do fluxo do algoritmo de detecção de objetos.

A imagem **C.1**, é gerada a partir dos *pixels* coincidentes entre a imagem **A.1** e a imagem **B.1**. Esta imagem conterá a parte estática do ambiente, portanto os dados de imagem **C.1** serão armazenados na memória. A 3ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.8.

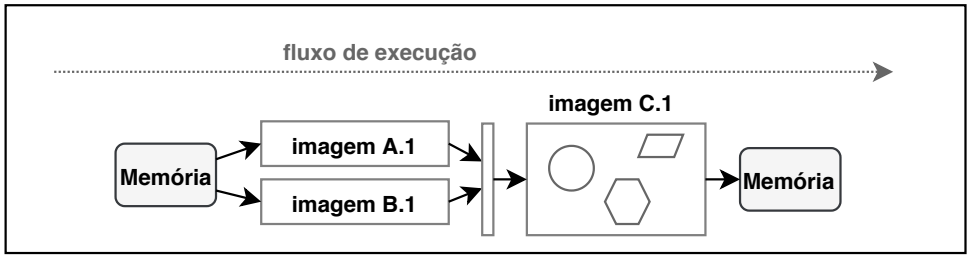


Figura 4.8: 3ª etapa do fluxo do algoritmo de detecção de objetos.

Após o tempo de espera, um novo quadro de vídeo **A.2** é capturado pela câmera

e os dados da imagem são armazenados na memória. O quadro de vídeo **A.2** é convertido do formato RGB, para escala de cinza, depois a detecção de borda é aplicada, as bordas (na cor preta) são armazenadas na memória. A 4ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.9.

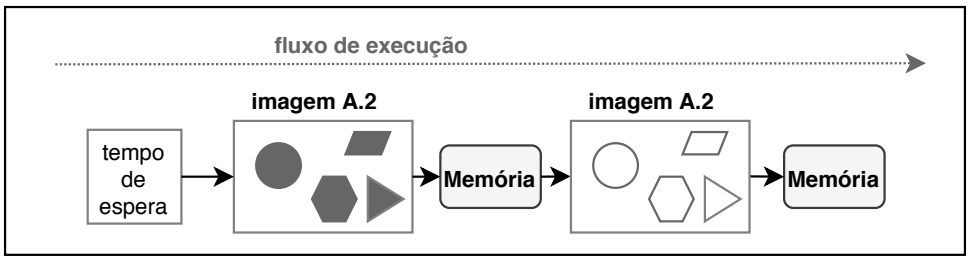


Figura 4.9: 4ª etapa do fluxo do algoritmo de detecção de objetos.

A imagem **B.2**, é gerada a partir dos *pixels* coincidentes entre as imagens **A.2** e **C.1**. Se houver uma diferença entre os *pixels* pretos de **A.2** e **C.1**, então há um novo objeto na imagem, e esses *pixels* devem ser armazenados na imagem **B.2**, os valores dos *pixels* devem ser incrementados por um valor pré-configurado. O final da 5ª etapa é um ponto referente ao tempo de espera, para obter um novo quadro de imagem, este será um ponto de retorno para o laço definido na 7ª etapa. A 5ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.10.

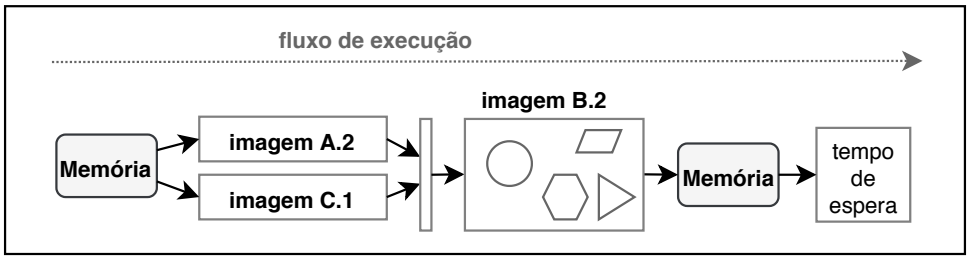


Figura 4.10: 5ª etapa do fluxo do algoritmo de detecção de objetos.

Após o tempo de espera, um novo quadro de vídeo **A.3** é capturado pela câmera, e os dados da imagem são armazenados na memória. O quadro de vídeo **A.3** é convertido do formato RGB, para escala de cinza, em seguida, a detecção de borda é aplicada e as bordas (na cor preta) são armazenadas na memória. A 6ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.11.

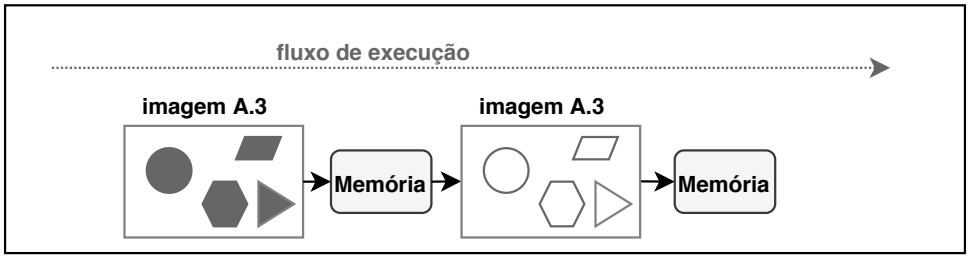


Figura 4.11: 6ª etapa do fluxo do algoritmo de detecção de objetos.

A imagem **B.2**, é atualizada a partir dos *pixels* coincidentes entre as imagens **A.3** e **C.1**. Se houver uma diferença entre os *pixels* pretos de **A.3** e **C.1**, então, há um novo objeto na imagem, e esses *pixels* devem ser armazenados na imagem **B.2**, os valores dos *pixels* devem ser incrementados por um valor pré-configurado. Além disso, na 7ª etapa é verificado se qualquer *pixel* da imagem **B.2**, atingiu um valor limite predefinido, no caso negativo, o sistema deve retornar para a 5ª etapa, para capturar um novo quadro de vídeo, repetindo o processo até que o limite seja atingido. Caso o limite predefinido tenha sido alcançado por qualquer *pixel* o sistema seguirá para a 8ª etapa. A 7ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.12.

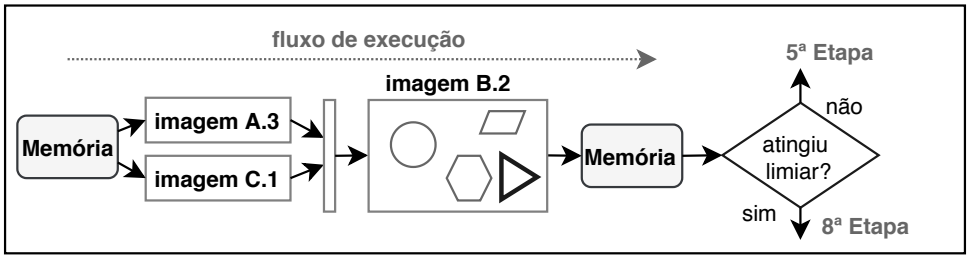


Figura 4.12: 7ª etapa do fluxo do algoritmo de detecção de objetos.

Quando qualquer *pixel* da imagem **B.2**, atinge o valor limite pré-definido, é realizada uma junção entre o último quadro de vídeo capturado **A.n** (**n** corresponde ao índice do último quadro de imagem capturado) e a imagem **B.2**, apenas para os *pixels* da imagem **B.2**, que foram incrementados. Finalmente, a imagem gerada com o objeto destacado é exibida e o sistema aguarda um sinal de *reset*. A 8ª etapa do fluxo do algoritmo de detecção de objetos pode ser visualizada na Figura 4.13. É importante destacar que apenas os *pixels* do objeto identificado são escritos na memória quando o limiar é atingido.

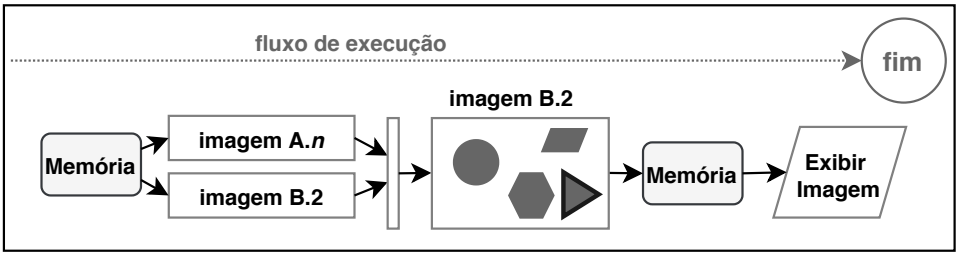


Figura 4.13: 8ª etapa do fluxo do algoritmo de detecção de objetos.

4.3 Desenvolvimento do Algoritmo de Detecção de Bordas

Na seção 4.2, foi apresentado o fluxograma que representa o procedimento de detecção de objetos em uma cena fixa, e foi possível notar que a função **Processamento de imagem** pode ser invocada toda vez que um novo quadro de vídeo de imagem é capturado. Tal função realiza o processo de detecção de bordas de uma imagem, e este processo exige um grande esforço computacional, devido a grande quantidade de operações lógicas e aritméticas realizadas. É importante ressaltar que esta função suporta diferentes resoluções de imagens. Nesta seção será apresentado o desenvolvimento da arquitetura, do algoritmo de detecção de bordas utilizado no projeto.

De acordo com o que foi abordado na seção 2.2.4, o algoritmo de detecção de bordas pode ser implementado através da convolução de uma máscara com uma imagem, como pode ser visto na Figura 4.14. A execução da máscara percorre toda a imagem.

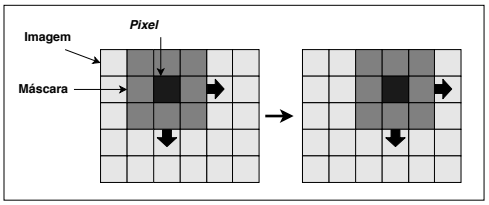


Figura 4.14: Convolução de uma máscara com uma imagem.

Como citado na seção 2.2.9, existem diferentes operadores que podem atuar como um detector de bordas. Para definir qual operador seria mais adequado para o projeto, foi realizada a implementação no software Matlab dos operadores de Roberts, Prewitt e Sobel. O resultado do processo de detecção de bordas utilizando os três

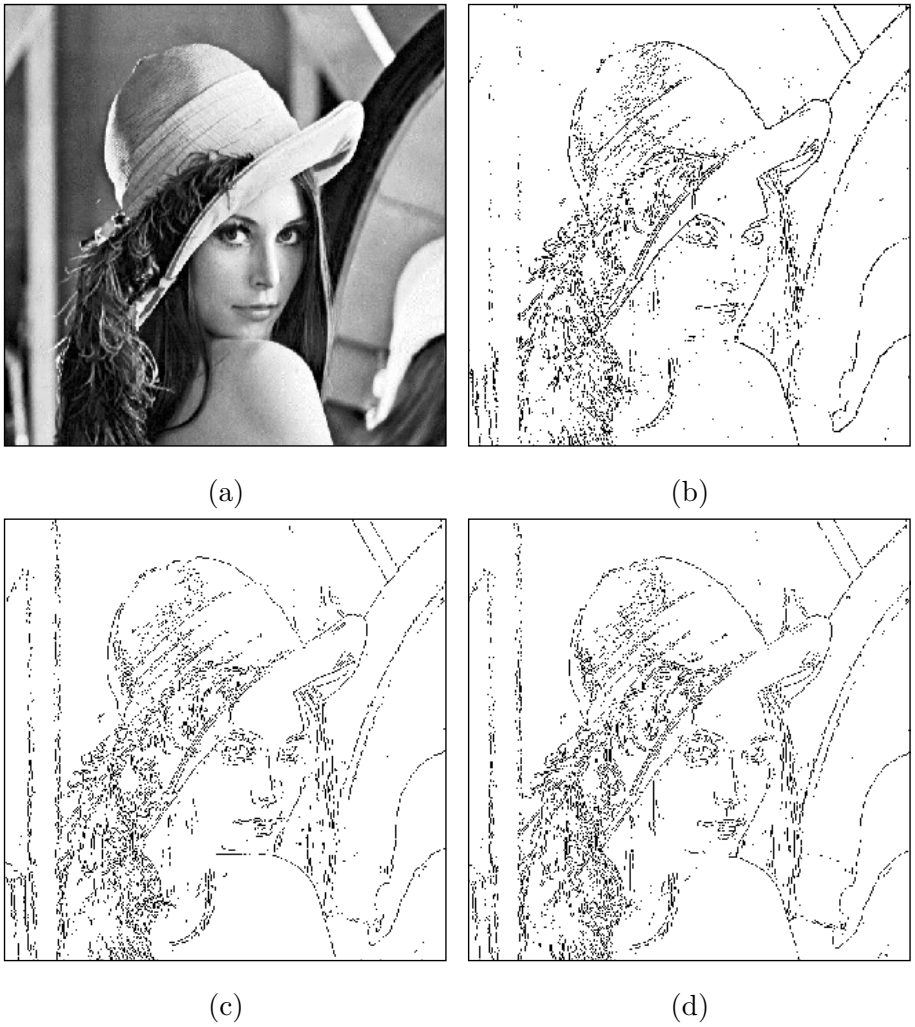


Figura 4.15: Resultado da aplicação dos operadores de Roberts, Prewitt e Sobel à uma imagem.

operadores pode ser visto na Figura 4.15. Na Figura 4.15a está localizada a imagem original utilizada no processamento, na Figura 4.15b está localizada a imagem resultante do processo de detecção de bordas, utilizando o operador de Roberts. Na Figura 4.15c está localizada a imagem resultante do processo de detecção de bordas, utilizando o operador de Prewitt. Por fim, na Figura 4.15d está localizada a imagem resultante do processo de detecção de bordas, utilizando o operador de Sobel.

O operador de Roberts utiliza uma máscara de tamanho $2x2$, proporcionando mais facilidade na implementação e mais rapidez na execução, em comparação aos operadores de Prewitt e Sobel. No entanto, o operador de Roberts é mais sensível aos ruídos na imagem, em comparação aos operadores de Prewitt e Sobel. O operador de Sobel é muito semelhante ao operador de Prewitt, ambos utilizam máscaras de

tamanho $3x3$, a diferença está localizada nos pesos da máscara, atribuindo maior peso aos pontos mais próximos do *pixel* central, e por esse motivo o operador de Sobel produz bordas diagonais menos atenuadas, em comparação com o operador de Prewitt. Portanto, com o intuito de obter maior precisão na detecção das bordas, neste trabalho foi utilizado um par de máscaras para a aproximação da magnitude do gradiente, chamado operadores de Sobel, que podem ser visualizados na máscara (4.1).

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.1}$$

A detecção de borda com a máscara de Sobel usa as duas máscaras citadas acima para efetuar a detecção. A máscara com coeficientes verticais diferentes de zero é usada para detectar derivadas horizontais da imagem. A máscara com o coeficiente horizontal diferente de zero é usada para detectar as derivadas verticais da imagem. Ambas as máscaras são iguais, exceto pela rotação de 90 graus. As duas máscaras, quando convoluídas com a imagem de entrada, produzem bordas nas direções X e Y. O resultado final é encontrado combinando a magnitude da direção X e Y.

Na Figura 4.16 é apresentado o fluxograma do algoritmo de detecção de bordas, **x** representa o número de linhas da imagem, e **y** representa o número de colunas da imagem. É importante ressaltar que, para efetuar o processamento de todos os *pixels*, convoluindo a máscara com a imagem, foi necessário adicionar bordas ao redor da imagem com o valor de intensidade do *pixel* igual a zero.

Como citado na seção 2.2.4, para cada *pixel* da imagem é calculado um novo *pixel* resultante em função do seu valor original e o valor dos *pixels* vizinhos, através da operação de convolução com a máscara de Sobel. No fluxograma apresentado na Figura 4.16, é possível visualizar as operações aritméticas efetuadas entre as máscaras de Sobel, as variáveis que armazenam o *pixel* a ser calculado e seus vizinhos.

Na Figura 4.17, é possível visualizar como ficam posicionadas as variáveis, que armazenam o *pixel* a ser calculado e seus vizinhos.

A arquitetura de hardware, para implementação do algoritmo de detecção de bordas, possui uma estrutura que inclui duas FIFO (*First In First Out*) e registradores de deslocamento. Cada *pixel* pode ser representado por um quadro da

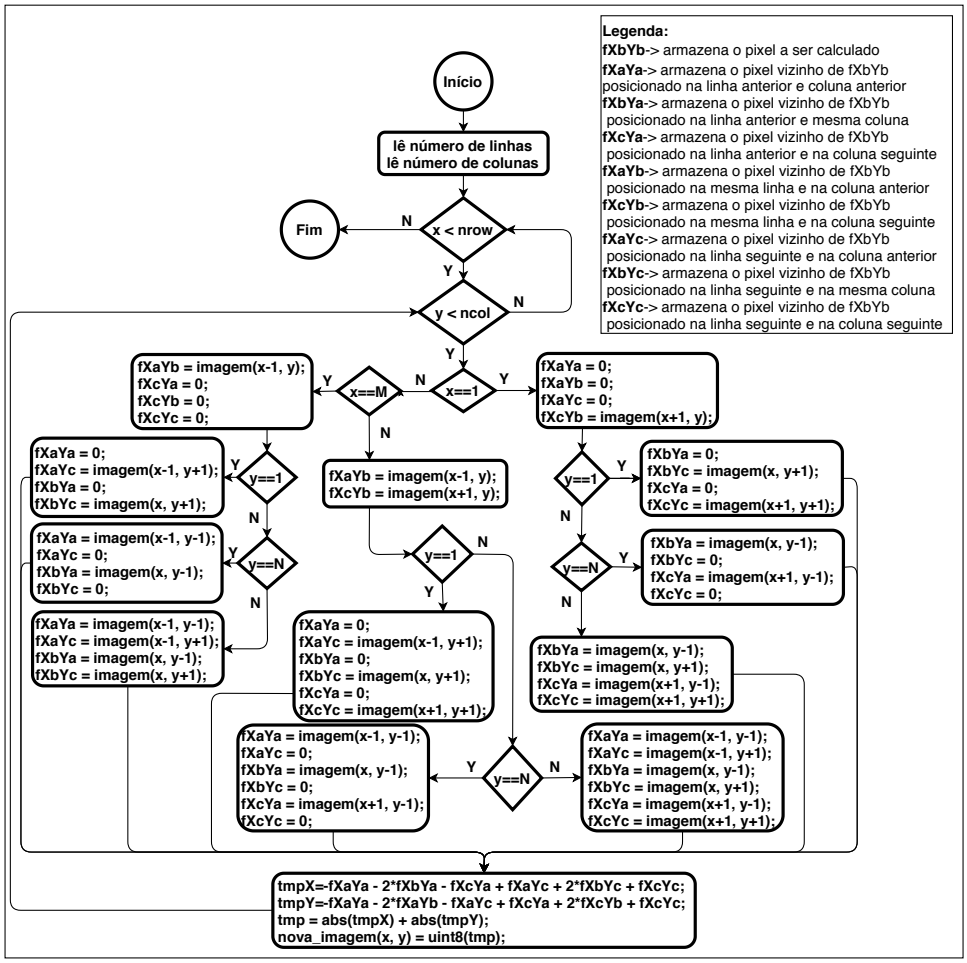


Figura 4.16: Fluxograma do algoritmo de detecção de bordas.

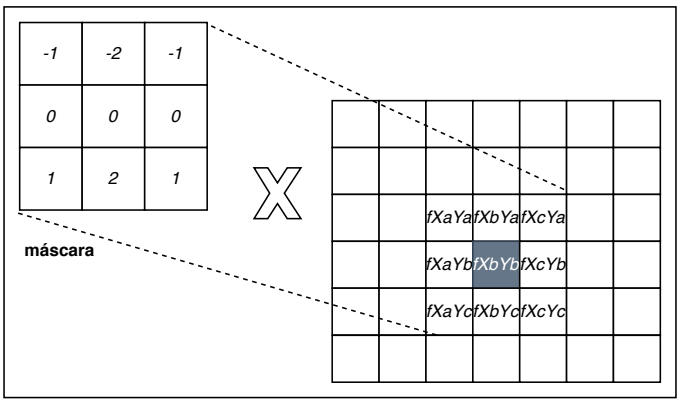


Figura 4.17: Variáveis que armazenam o *pixel* a ser calculado e seus vizinhos.

imagem. A cada ciclo de *clock*, os *pixels* são deslocados na mesma direção, com o mais antigo sendo removido. Cada *pixel* armazenado nas FIFO ou nos registradores de deslocamento, representa uma linha da imagem. O tamanho das FIFO é definido pelo número de colunas na imagem, e o tamanho da máscara usado.

Como pode ser visualizado na Figura 4.18, a leitura dos *pixels* é realizada no

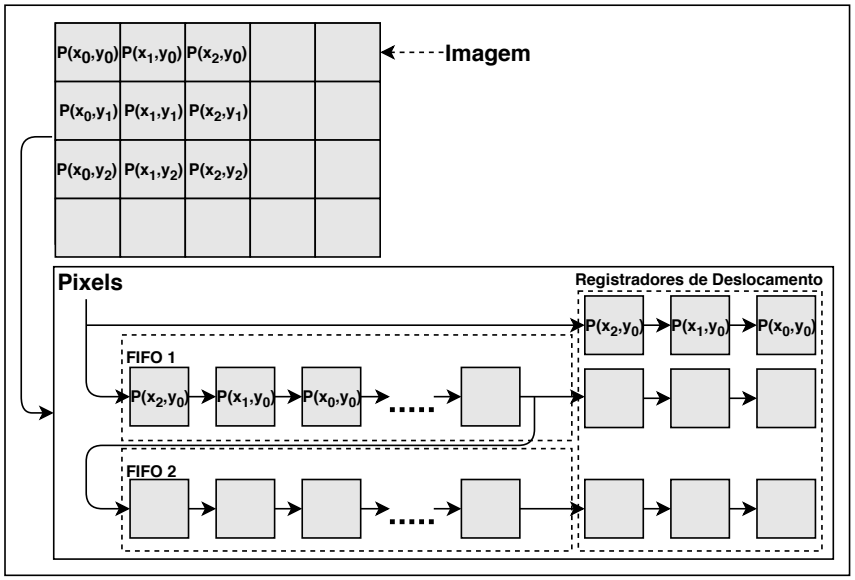


Figura 4.18: Arquitetura de hardware para o algoritmo de detecção de borda.

sentido da esquerda para a direita, iniciando a partir do canto superior esquerdo da imagem. Os *pixels* são inseridos em paralelo na entrada da FIFO 1 e na entrada do primeiro grupo de registradores de deslocamento, a cada ciclo de *clock* os *pixels* são deslocados na mesma direção. Quando a FIFO 1 está cheia os *pixels* mais antigos são inseridos em paralelo na entrada da FIFO 2 e na entrada do segundo grupo de registradores de deslocamento. Na última etapa do fluxo, quando a FIFO 2 está cheia os *pixels* mais antigos são inseridos no terceiro grupo de registradores deslocamento. Desta forma é possível obter o *pixel* central e seus vizinhos para realizar o cálculo com a máscara de coeficientes $3x3$.

Com base na arquitetura da Figura 4.18, é possível obter os *pixels* calculados, a partir de operações de multiplicação e adição entre os coeficientes de máscara e os *pixels* vizinhos na região de interesse.

4.4 Projeto da Arquitetura do Algoritmo de Comparação entre Quadros de Imagens Digitais

Como foi visto na seção 2.5, na abordagem *Background subtraction*, os objetos a serem detectados são subtraídos *pixel* por *pixel*, de uma imagem de fundo estática. A imagem de fundo estática, ou modelo, é criada calculando a média das imagens ao longo do tempo. Foi visto também a abordagem *Frame differencing*, que é baseada

na detecção de objetos em movimento, calculando a diferença *pixel* por *pixel*, de dois quadros consecutivos em uma sequência de vídeo. Essa diferença é então comparada a um limiar, para determinar se um objeto está na imagem de fundo ou em primeiro plano. A técnica *Temporal Differencing* (diferenciação temporal) detecta o objeto em movimento, empregando um método de diferença de *pixel* em quadros sucessivos. Também foi apresentado o método *Optical Flow*, no entanto neste trabalho foram usados apenas conceitos abordados nos métodos *Frame differencing*, *Temporal Differencing* e *Background subtraction*.

No início do processo de detecção de objetos, foram usados alguns conceitos baseados no método *Background subtraction*, onde a imagem de fundo estática é gerada a partir da comparação entre quadros de imagem obtidos ao longo do tempo. Após a geração da imagem de fundo estática, são usados conceitos baseados nos métodos *Frame differencing*, *Temporal Differencing*. Um novo quadro de imagem é obtido, e o sistema realiza a comparação entre o quadro obtido e a imagem de fundo estática, com o objetivo de gerar um novo quadro de imagem, que será utilizado como referência. Quando os valores dos *pixels* do quadro de referência atingirem um valor de limiar, será possível determinar se o objeto está na imagem de fundo ou em primeiro plano. Ao longo do tempo de execução do algoritmo, novos quadros de imagem são obtidos e comparados ao quadro de referência, e toda vez o valor do limiar é testado, para definir se o objeto foi detectado. Na Figura 4.19, pode ser visualizado o fluxograma do algoritmo de comparação entre quadros de imagens, implementado no projeto.

Na Figura 4.19 são apresentadas três operações, que representam as etapas do algoritmo de comparação entre quadros de imagens. O controle principal do sistema sinalizará que a primeira operação realizada, será a **Geração da imagem estática**. Esta operação realiza a leitura simultânea de dois quadros de imagem, na memória SDRAM e compara *pixel* a *pixel* entre as duas imagens. O resultado da comparação entre os *pixels* é um novo valor para o *pixel* de um novo quadro de imagem, que será armazenado na memória SDRAM. Em seguida, o controle principal do sistema sinalizará que a segunda operação realizada, será a **Geração da imagem auxiliar**. Esta operação realiza a leitura simultânea de dois quadros de imagem da memória SDRAM, um quadro refere-se à uma nova imagem capturada pela câmera, e o outro

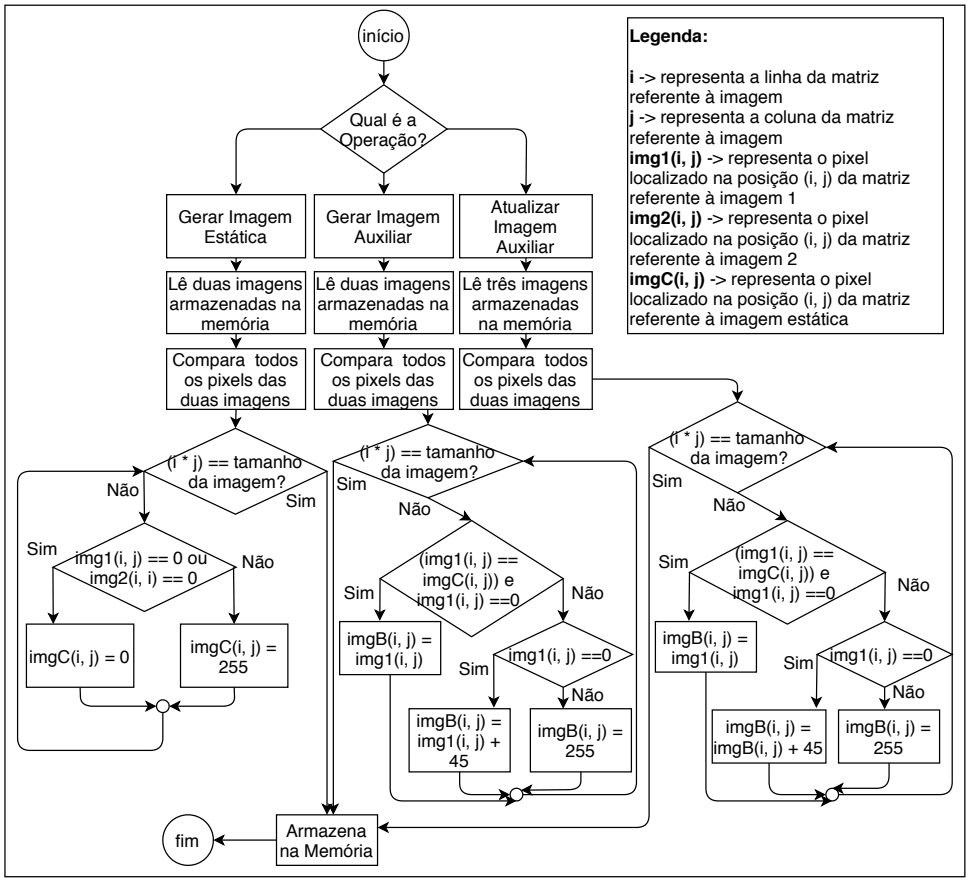


Figura 4.19: Fluxograma do algoritmo de comparação entre quadros de imagens digitais.

quadro refere-se à imagem estática, resultante da operação de geração da imagem estática. Esta operação também realiza a comparação *pixel* a *pixel* entre as duas imagens, e o resultado da comparação entre os *pixels* é um novo valor para o *pixel* de um novo quadro de imagem, que será armazenado na memória SDRAM.

Por fim, o controle principal do sistema sinalizará que a primeira operação realizada, será a **Atualização da imagem auxiliar**. Esta operação realiza a leitura simultânea de três quadros de imagem da memória SDRAM, o primeiro quadro refere-se à uma nova imagem capturada pela câmera, o segundo quadro refere-se à imagem estática, resultante da operação de **Geração da imagem estática**, e o terceiro quadro refere-se à imagem auxiliar, resultante da operação de **Geração da imagem auxiliar**. Esta operação também realiza a comparação *pixel* a *pixel* entre as três imagens, e o resultado da comparação entre os *pixels* é um novo valor para o *pixel* do quadro da imagem auxiliar, que será armazenado na memória SDRAM.

Capítulo 5

Arquitetura de Hardware Desenvolvida

A arquitetura de hardware proposta neste trabalho, deve ser capaz de realizar a detecção de objetos, bem como identificá-los em tempo real. Para que isso seja possível, os módulos de processamento desenvolvidos tem a restrição de operar na frequência de 50 MHz, como será visto na seção 7.1.

Neste capítulo são apresentados os diagramas de blocos, com os sinais de entrada e saída, bem como a lógica de controle de todos os módulos implementados no projeto. Também são apresentados detalhes sobre as principais funções de cada bloco.

5.1 Visão Geral da Arquitetura do Sistema

Neste trabalho, a Linguagem de Descrição de Hardware Verilog foi usada para modelar o hardware. A Arquitetura proposta consiste de 11 módulos implementados, que são: FSM Control, Input Handler Ctrl, RGB2GRAY Ctrl, CONV RGB2GRAY, EDGE DETECT CTRL, EDGE DETECT, EDGE STORAGE, Compare Pixel CTRL, Counter, MUX_WR_SDRAM e MUX_RD_SDRAM. Os módulos IP (*Intellectual Property*) que compõem a arquitetura são: CCD Capture, I2C CCD CONFIG, VGA CTRL, RAW2RGB e SDRAM CTRL. Os módulos IP foram configurados de acordo com os requisitos do projeto. Na Figura 5.1 é possível visualizar a arquitetura geral do sistema.

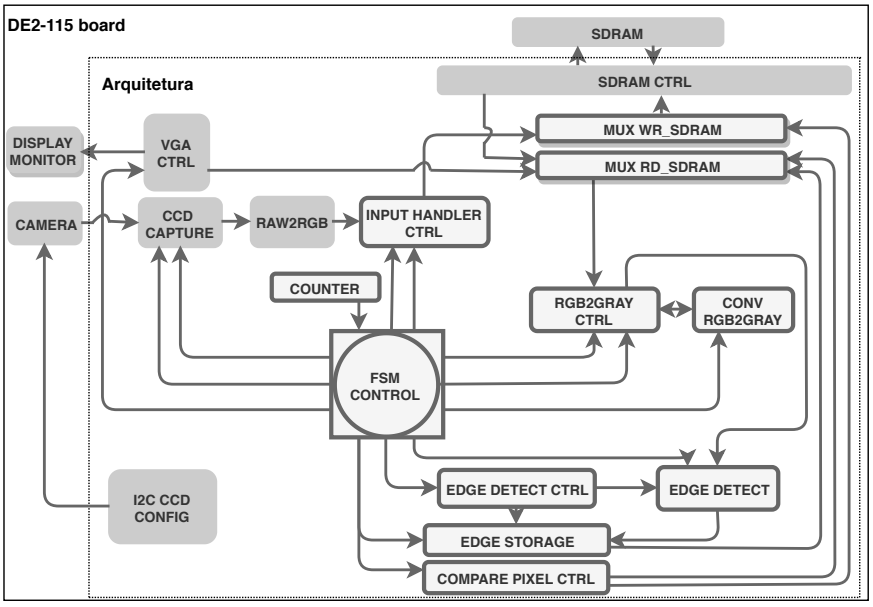


Figura 5.1: Arquitetura de hardware.

O módulo FSM Control é uma máquina de estados finita, que coordena as operações entre todos os módulos do sistema. Este bloco recebe e gera sinais de controle que reinicializam, inicializam e executam operações em todos os módulos de processamento.

O módulo Input Handler Ctrl lê os dados do quadro de vídeo do módulo RAW2RGB (IP) converte os *pixels* recebidos no tamanho de 8 bits e grava os dados no SDRAM.

O módulo RGB2GRAY Ctrl lê os dados do quadro de vídeo da SDRAM, organiza e envia para o módulo CONV RGB2GRAY. O módulo CONV RGB2GRAY converte os *pixels* da imagem no formato RGB para tons de cinza.

O módulo EDGE DETECT CTRL coordena as operações entre os módulos EDGE DETECT e EDGE STORAGE. Além disso, adiciona uma borda aos dados do quadro de vídeo, antes do processamento da detecção de bordas. O módulo EDGE DETECT implementa o algoritmo de detecção de borda, usando o operador Sobel e pré-configurando um limiar como um parâmetro. O módulo EDGE STORAGE lê dados processados pelo módulo EDGE DETECT, organiza e grava dados no SDRAM.

O módulo Compare Pixel CTRL implementa o algoritmo de detecção de objeto em uma cena fixa. Este bloco lê / grava dados no SDRAM.

O módulo CCD Capture é um *Soft IP Core* que extrai *pixels* válidos da interface com a câmera. O módulo RAW2RGB é um *Soft IP Core* que converte dados da

câmera no formato *Bayer Pattern* para o formato RGB. O módulo I2C CCD CONFIG é um *Soft IP Core* que configura os parâmetros da câmera através de registros de configuração.

O módulo VGA CTRL é um *Soft IP Core* que lê dados de um quadro de vídeo da memória SDRAM e sincroniza dados com a interface VGA.

O módulo Counter é um componente responsável por implementar um contador de 32 bits, e os módulos MUX_WR_SDRAM e MUX_RD_SDRAM são componentes que implementam multiplexadores, que auxiliam na seleção de endereços de escrita e leitura para a memória SDRAM.

5.1.1 CCD Capture

Este módulo é responsável por receber os dados advindos da câmera CMOS. A função essencial deste bloco é extrair os dados de *pixels* válidos, levando em consideração as informações dos sinais FVAL (*Frame Valid*) e LVAL (*Line Valid*) da câmera CMOS. A frequência do *clock* deste módulo é de 50 MHz. O sinal de saída DVAL indica se os dados de *pixel* sob análise são válidos ou não.

Este módulo conta o número de quadros levando em consideração o sinal FVAL, e mostra a contagem no *display* de sete segmentos do dispositivo FPGA. As coordenadas X e Y de cada *pixel* válido de uma imagem são usadas como entradas para o módulo RAW2RGB, e indicam uma contagem dos *pixels* até atingir a resolução da imagem configurada. As entradas START e END determinam o início e a pausa do processamento, respectivamente, ambos podem ser controlados pelo módulo FSM Ctrl ou através dos *push buttoms* do dispositivo FPGA. Este módulo recebe como parâmetro de entrada a resolução da imagem, que pode ser $800x600$ ou $640x480$. Na Figura 5.2 pode-se visualizar o diagrama de blocos do módulo CCD Capture.

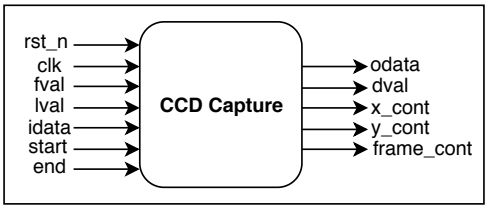


Figura 5.2: Módulo CCD capture.

5.1.2 RAW2RGB

Este módulo recebe os dados obtidos pelo módulo CCD Capture e direciona o fluxo de dados para o módulo Input Handler Ctrl. Este módulo converte os dados do formato *Bayer Pattern*, advindo da câmera CMOS para o formato RGB. O método adotado consiste em extrair quatro *pixels* adjacentes em um quadrado (R, G1, G2, B) e converter para um *pixel* no formato RGB.

As coordenadas X e Y de cada *pixel* válido de uma imagem indicam uma contagem dos *pixels*, até atingir a resolução da imagem configurada. A entrada iDVAL indica quando aparece um dado válido para ser capturado, e a saída oDVAL indica quando um dado está disponível após o processamento. Na Figura 5.3 pode-se visualizar o diagrama de blocos do módulo RAW2RGB.

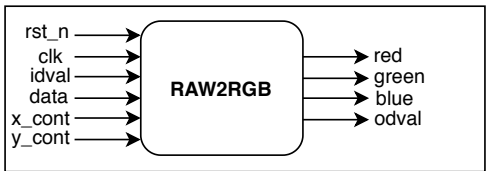


Figura 5.3: Módulo RAW2RGB.

5.1.3 I2C CCD Config

A câmera sensor CMOS suporta uma interface serial, baseada no protocolo de barramento I2C. Este módulo usa uma interface serial para configurar os registradores da câmera sensor CMOS. O barramento I2C é composto de duas linhas bidirecionais, SDA (*Serial Data*) e SCL (*Serial Clock*). Cada dispositivo no barramento I2C, tem um endereço exclusivo e pode operar como um emissor ou como um receptor. Um dispositivo atua como mestre e os outros atuam como escravos. O dispositivo mestre gera o sinal de *clock* no SCL e inicia/termina a transferência de dados. A transferência de dados ocorre através do SDA.

Através dos registradores de configuração, é possível modificar os parâmetros essenciais de formatação e temporização da câmera, usadas para a geração das imagens. Por exemplo, é possível alterar o *frame rate*, bem como aumentar ou diminuir, o brilho de uma imagem capturada pela câmera. Na Figura 5.4, pode-se visualizar o diagrama de blocos do módulo I2C CCD Config.

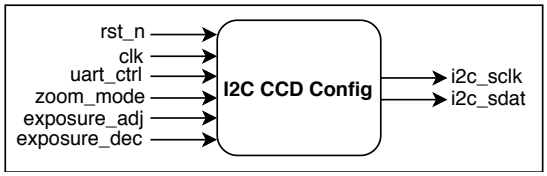


Figura 5.4: Módulo I2C CCD config.

5.1.4 Input Handler Ctrl

Este módulo é o responsável por direcionar os dados, advindos do módulo RAW2RGB, para a memória SDRAM, onde serão armazenados. A câmera CMOS fornece 3 canais de cores de 12 bits cada. No entanto, o formato RGB (*true color*) necessita apenas de 8 bits por canal para representar o *pixel*. Portanto, este módulo visa a remoção dos 4 bits menos significativos de cada dado que representa um canal de cor (R, G, B), para o posterior armazenamento na memória SDRAM.

Outra função relevante deste módulo é o monitoramento na contagem dos quadros, para indicar quando se tem o início de um quadro de imagem válido. A quantidade de *pixels* recebidos também é monitorada, de forma que ao atingir a resolução da imagem pré-configurada, o sinal wr_done é ativado. Na Figura 5.5 pode-se visualizar o diagrama de blocos do módulo Input Handler Ctrl.

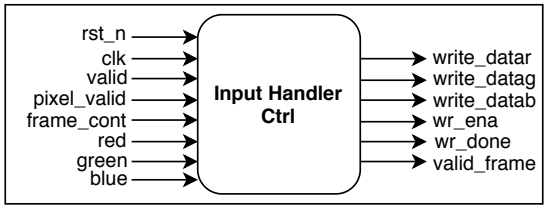


Figura 5.5: Módulo input handler ctrl.

5.1.4.1 Lógica de Controle do módulo Input Handler Ctrl

Este módulo é composto por uma máquina de estados finita do tipo *Mealy* que possui 5 estados, Figura 5.6.

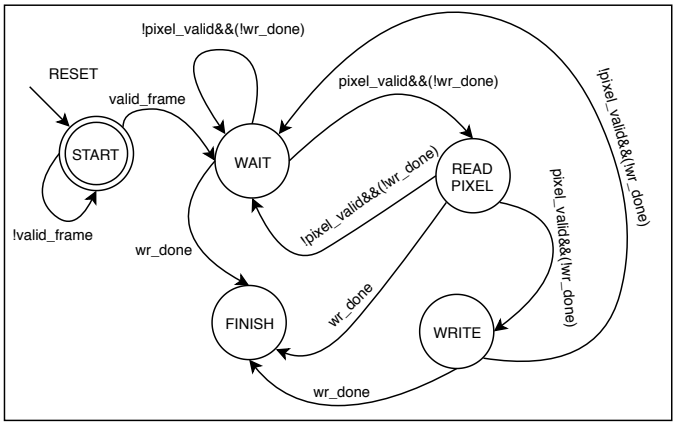


Figura 5.6: FSM - input handler ctrl.

5.1.5 FSM Control

Este módulo é responsável por coordenar as operações entre todos os outros módulos do sistema. Por esse motivo, existe uma grande quantidade de sinais de entrada e saída, as entradas são utilizadas pela lógica de controle para efetuar as transições entre os estados, e as saídas habilitam funções e operações dos outros módulos.

5.1.5.1 Lógica de Controle do módulo FSM Control

Este módulo é composto por uma máquina de estados finita, do tipo *Mealy* que possui 39 estados. Nesta seção serão apresentados os estados, entradas e funções de saída para cada estado, com o objetivo de demonstrar como a lógica de controle principal, coordena as operações para todo o sistema. Na descrição dos estados, são feitas referências aos quadros de imagens **A.1**, **B.1**, **C.1**, **A.2**, **B.2** e **A.n**, citados na descrição do algoritmo de detecção de objetos na seção 4.2.

Descrição dos Estados:

- START - Entradas: O sinal valid indica que o sistema foi iniciado. Função de Saída: Enviar sinal de reset para todos os blocos. Iniciar processo para capturar quadro **A.1**. Próximo Estado: CAPTURE_IMAGE_A ou START.
- CAPTURE_IMAGE_A - Função de Saída: Configurar os endereços para o armazenamento do quadro **A.1** na memória SDRAM. Próximo Estado: WRITE_IMAGE_A.
- WRITE_IMAGE_A - Entradas: O sinal input_ctrl_done indica que o módulo

Input Handler Ctrl finalizou a captura de um quadro. Função de Saída: Enviar um sinal indicando para o módulo CCD Capture para obter um novo quadro de imagem. Próximo Estado: WRITE_IMAGE_A ou READ_IMAGE_A.

- READ_IMAGE_A - Função de Saída: Configurar os endereços para a leitura do quadro **A.1** na memória SDRAM pelo módulo RGB2GRAY CTRL. Próximo Estado: FILTER_IMAGE_A.

- FILTER_IMAGE_A - Entradas: O sinal edge_store_done indica que o processo de detecção de bordas foi finalizado, e o quadro **A.1** armazenado na memória SDRAM. Função de Saída: Inicializar os módulos RGB2GRAY CTRL e EDGE DETECT, que converte a imagem de RGB para escala de cinza e realiza o processo de detecção de bordas, respectivamente. Próximo Estado: FILTER_IMAGE_A ou READ_EDGE_A.

- READ_EDGE_A - Função de Saída: Configurar os endereços para a leitura do quadro **A.1** (com as bordas detectadas) na memória SDRAM pelo módulo COMPARE_PIXEL_CTRL. Próximo Estado: DISPLAY_ADJUST.

- DISPLAY_ADJUST - Entradas: O sinal cmp_pixel_done indica que o armazenamento do quadro **A.1** (com as bordas detectadas) foi realizado no espaço de endereçamento do módulo VGA_CTRL. Função de Saída: Inicializar o módulo COMPARE_PIXEL_CTRL, que lê o quadro **A.1** (com as bordas detectadas) e armazena no espaço de endereçamento do módulo VGA_CTRL. Próximo Estado: DISPLAY_ADJUST ou READ_RGB_A.

- READ_RGB_A - Função de Saída: Configurar os endereços para a leitura do quadro **A.1** (com as bordas detectadas) na memória SDRAM pelo módulo VGA_CTRL. Próximo Estado: DISPLAY_IMAGE_A.

- DISPLAY_IMAGE_A - Entradas: O sinal vga_ctrl_done indica que a exibição do quadro **A.1** (com as bordas detectadas) no Monitor VGA foi realizada. Função de Saída: Inicializar o módulo VGA_CTRL que lerá os dados do quadro **A.1** (com as bordas detectadas). Próximo Estado: DISPLAY_IMAGE_A ou WAIT_TIME_0.

- WAIT_TIME_0 - Entradas: O sinal counter_done indica que o módulo Counter finalizou a contagem até um valor predefinido. Função de Saída: Inicializar o módulo Counter, que tem a função de simular um estado de espera durante um período predefinido. Próximo Estado: CAPTURE_IMAGE_B ou WAIT_-TIME_0.

- CAPTURE_IMAGE_B - Função de Saída: Configurar os endereços para o armazenamento do quadro **B.1** na memória SDRAM. Próximo Estado: WRITE_-IMAGE_B.

- WRITE_IMAGE_B - Entradas: O sinal input_ctrl_done indica que o módulo Input Handler Ctrl finalizou a captura de um quadro. Função de Saída: Enviar um sinal indicando para o módulo CCD Capture para obter um novo quadro de imagem. Próximo Estado: WRITE_IMAGE_B ou READ_IMAGE_B.

- READ_IMAGE_B - Função de Saída: Configurar os endereços para a leitura do quadro **B.1** na memória SDRAM pelo módulo RGB2GRAY CTRL. Próximo Estado: FILTER_IMAGE_B.

- FILTER_IMAGE_B - Entradas: O sinal edge_store_done indica que o processo de detecção de bordas foi finalizado, e o quadro **B.1** armazenado na memória SDRAM. Função de Saída: Inicializar os módulos RGB2GRAY CTRL e EDGE DETECT, que converte a imagem de RGB para escala de cinza e realiza o processo de detecção de bordas, respectivamente. Próximo Estado: FILTER_-IMAGE_B ou READ_EDGE_B.

- READ_EDGE_AB - Função de Saída: Configurar os endereços para a leitura dos quadros **A.1** e **B.1** (com as bordas detectadas) na memória SDRAM pelo módulo COMPARE_PIXEL_CTRL. Próximo Estado: GENERATE_IMAGE_-C.

- GENERATE_IMAGE_C - Entradas: O sinal cmp_pixel_done indica que o processo de comparação entre os quadros **A.1** e **B.1** foi finalizado, e o quadro **C.1** armazenado na memória SDRAM. Função de Saída: Inicializar o módulo COMPARE_PIXEL_CTRL, que realiza a comparação entre os quadros **A.1** e

B.1, gerando assim o quadro **C.1**. Próximo Estado: GENERATE_IMAGE_C ou CAPTURE_NEW_IMG_A.

- CAPTURE_NEW_IMG_A - Função de Saída: Configurar os endereços para o armazenamento do quadro **A.2** na memória SDRAM. Próximo Estado: WRITE_NEW_IMG_A.

- WRITE_NEW_IMG_A - Entradas: O sinal input_ctrl_done indica que o módulo Input Handler Ctrl finalizou a captura de um quadro. Função de Saída: Enviar um sinal indicando para o módulo CCD Capture para obter um novo quadro de imagem. Próximo Estado: WRITE_NEW_IMG_A ou READ_NEW_IMG_A.

- READ_NEW_IMG_A - Função de Saída: Configurar os endereços para a leitura do quadro **A.2** na memória SDRAM pelo módulo RGB2GRAY CTRL. Próximo Estado: FILTER_NEW_IMG_A.

- FILTER_NEW_IMG_A - Entradas: O sinal edge_store_done indica que o processo de detecção de bordas foi finalizado, e o quadro **A.2** armazenado na memória SDRAM. Função de Saída: Inicializar os módulos RGB2GRAY CTRL e EDGE DETECT, que converte a imagem de RGB para escala de cinza e realiza o processo de detecção de bordas, respectivamente. Próximo Estado: FILTER_NEW_IMG_A ou READ_EDGE_AC.

- READ_EDGE_AC - Função de Saída: Configurar os endereços para a leitura dos quadros **A.2** e **C.1** (com as bordas detectadas) na memória SDRAM pelo módulo COMPARE_PIXEL_CTRL. Próximo Estado: GENERATE_IMAGE_B.

- GENERATE_IMAGE_B - Entradas: O sinal cmp_pixel_done indica que o processo de comparação entre os quadros **A.2** e **C.1** foi finalizado, e o quadro **B.2** armazenado na memória SDRAM. Função de Saída: Inicializar o módulo COMPARE_PIXEL_CTRL, que realiza a comparação entre os quadros **A.2** e **C.1**, gerando assim o quadro **B.2**. Próximo Estado: GENERATE_IMAGE_B ou WAIT_TIME_1.

- WAIT_TIME_1 - Entradas: O sinal counter_done indica que o módulo Counter finalizou a contagem até um valor predefinido. Função de Saída: Inicializar o

módulo Counter, que tem a função de simular um estado de espera durante um período predefinido. Próximo Estado: CAPTURE_IMG2_A ou WAIT_TIME_1.

- CAPTURE_IMG2_A - Função de Saída: Configurar os endereços para o armazenamento do quadro **A.n** na memória SDRAM. Próximo Estado: WRITE_IMG2_A.

- WRITE_IMG2_A - Entradas: O sinal input_ctrl_done indica que o módulo Input Handler Ctrl finalizou a captura de um quadro. Função de Saída: Enviar um sinal indicando para o módulo CCD Capture para obter um novo quadro de imagem. Próximo Estado: WRITE_IMG2_A ou READ_IMG2_A.

- READ_IMG2_A - Função de Saída: Configurar os endereços para a leitura do quadro **A.n** na memória SDRAM pelo módulo RGB2GRAY CTRL. Próximo Estado: FILTER_IMG2_A.

- FILTER_IMG2_A - Entradas: O sinal edge_store_done indica que o processo de detecção de bordas foi finalizado, e o quadro **A.n** armazenado na memória SDRAM. Função de Saída: Inicializar os módulos RGB2GRAY CTRL e EDGE DETECT, que converte a imagem de RGB para escala de cinza e realiza o processo de detecção de bordas, respectivamente. Próximo Estado: FILTER_IMG2_A ou READ_EDGE_ABC.

- READ_EDGE_ABC - Função de Saída: Configurar os endereços para a leitura dos quadros **A.n**, **B.2** e **C.1** (com as bordas detectadas) na memória SDRAM pelo módulo COMPARE_PIXEL_CTRL. Próximo Estado: UPDATE_IMAGE_B.

- UPDATE_IMAGE_B - Entradas: O sinal cmp_pixel_done indica que o processo de comparação entre os quadros **A.n**, **B.2** e **C.1** foi finalizado, e o quadro **B.2** armazenado na memória SDRAM. Função de Saída: Inicializar o módulo COMPARE_PIXEL_CTRL, que realiza a comparação entre os quadros **A.n**, **B.2** e **C.1**, atualizando assim o quadro **B.2**. Próximo Estado: UPDATE_IMAGE_B ou CHECK_THRESHOLD.

- CHECK_THRESHOLD - Entradas: O sinal threshold_ena indica que algum pixel atingiu o limiar, e o objeto foi detectado. Função de Saída: Reinicializar as FIFOs de armazenamento. Próximo Estado: READ_B_GENERATED ou WAIT_TIME1.

- READ_B_GENERATED - Função de Saída: Configurar os endereços para a leitura do quadro **B.2** na memória SDRAM pelo módulo COMPARE_PIXEL_CTRL. Próximo Estado: WRITE_IMG_B.

- WRITE_IMG_B - Entradas: O sinal cmp_pixel_done indica que o quadro **B.2** foi armazenado num espaço de endereçamento para realização da junção dos quadros. Função de Saída: Inicializar o módulo COMPARE_PIXEL_CTRL, que obtém o quadro **B.2** e armazena no espaço de endereçamento correspondente para a realização da junção entre os quadros. Próximo Estado: READ_B_FINAL.

- READ_B_FINAL - Função de Saída: Configurar os endereços para a leitura dos quadros **B.2** e **A.n** na memória SDRAM pelo módulo COMPARE_PIXEL_CTRL. Próximo Estado: PREMERGE_IMAGE.

- PREMERGE_IMAGE - Entradas: O sinal cmp_pixel_done indica que o quadro **B.2** e **A.n** foram armazenados num espaço de endereçamento correspondente para a realização da junção dos quadros. Função de Saída: Inicializar o módulo COMPARE_PIXEL_CTRL, que obtém o quadro **B.2** e **A.n** e armazena no espaço de endereçamento correspondente para a realização da junção entre os quadros. Próximo Estado: READ_B_A_FINAL.

- READ_B_A_FINAL - Função de Saída: Configurar os endereços para a leitura dos quadros **B.2** e **A.n** na memória SDRAM pelo módulo COMPARE_PIXEL_CTRL. Próximo Estado: MERGE_IMAGE.

- MERGE_IMAGE - Entradas: O sinal cmp_pixel_done indica que o quadro **B.2** e **A.n** estão disponíveis num determinado espaço de endereçamento. Função de Saída: Inicializar o módulo COMPARE_PIXEL_CTRL, que obtém o quadro **B.2** e **A.n** para a realização da junção dos quadros. Próximo Estado: READ_IMAGE_FINAL.

- READ_IMAGE_FINAL - Função de Saída: Configurar os endereços para a leitura do quadro de imagem (com o objeto identificado) na memória SDRAM pelo módulo VGA_CTRL. Próximo Estado: VIEW_IMAGE.

- VIEW_IMAGE - Entradas: O sinal vga_ctrl_done indica que o quadro de imagem (com o objeto identificado) foi lido pelo módulo VGA_CTRL. Função de Saída: Inicializar o módulo VGA_CTRL, que obtém o quadro de imagem (com o objeto identificado). Próximo Estado: FINISH.

- FINISH - Entradas: O sinal rst_n indica se o sistema deve ser reinicializado. Função de Saída: Mantém o módulo VGA_CTRL realizando a leitura do quadro de imagem (com o objeto identificado). Próximo Estado: FINISH.

A máquina de estados que representa este módulo pode ser visualizada na Figura 5.7.

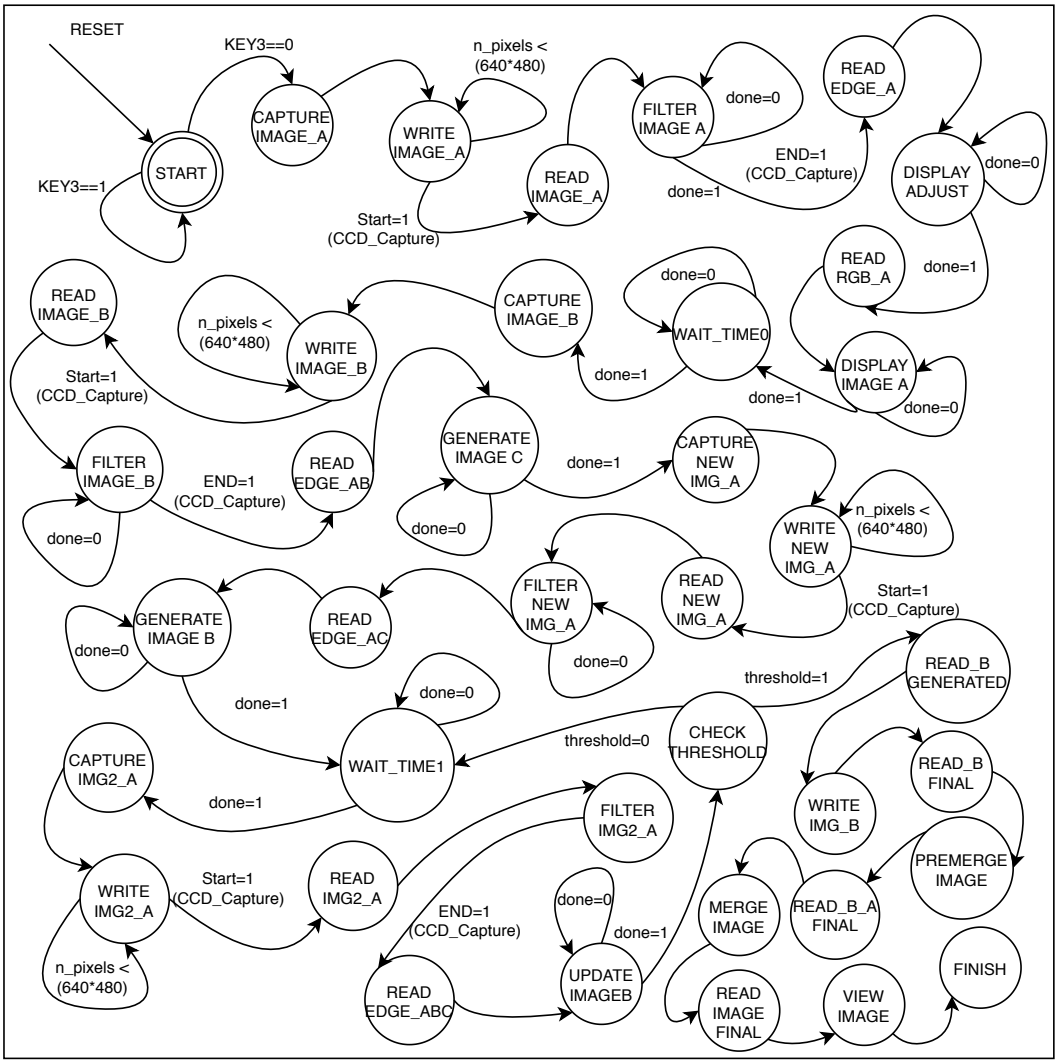
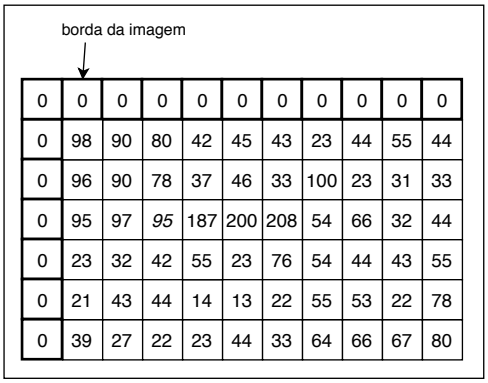


Figura 5.7: FSM - control.

5.1.6 EDGE DETECT CTRL

Este módulo é responsável por coordenar as operações entre os módulos EDGE DETECT e EDGE STORAGE. Ele recebe os dados convertidos, do formato RGB para escala de cinza, e realiza um processo referente a adição das bordas da imagem, como pode ser visto na Figura 5.8. Como foi explicado na seção 4.3, este passo é fundamental para que o tamanho da imagem continue com a resolução $800x600$ ou $640x480$. Para realizar este processo é necessária a utilização de uma FIFO, para armazenar os *pixels* da imagem no momento em que os *pixels* da borda da imagem são adicionados.

Este módulo monitora toda o processo de detecção de bordas, informando ao módulo EDGE STORAGE quando um pixel com a borda detectada está disponível.



0	0	0	0	0	0	0	0	0	0	0
0	98	90	80	42	45	43	23	44	55	44
0	96	90	78	37	46	33	100	23	31	33
0	95	97	*95*	187	200	208	54	66	32	44
0	23	32	42	55	23	76	54	44	43	55
0	21	43	44	14	13	22	55	53	22	78
0	39	27	22	23	44	33	64	66	67	80

Figura 5.8: Adição das bordas da imagem.

Um protocolo, entre este módulo e o módulo EDGE DETECT, refere-se à disponibilidade do EDGE DETECT para receber um novo dado. Isto é feito através do sinal buffer0_full, que indica se a FIFO interna do módulo EDGE DETECT está cheia ou não. Na Figura 5.9 é mostrado o diagrama de blocos do módulo EDGE DETEC CTRL.

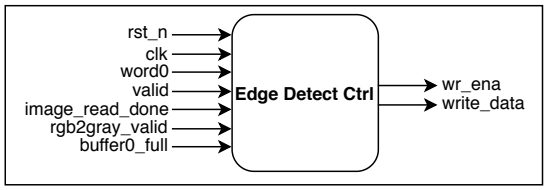


Figura 5.9: Módulo edge detect ctrl.

5.1.6.1 Lógica de Controle do módulo Edge Detect Ctrl

Este módulo é composto por uma máquina de estados finita do tipo *Mealy* que possui 10 estados, Figura 5.10.

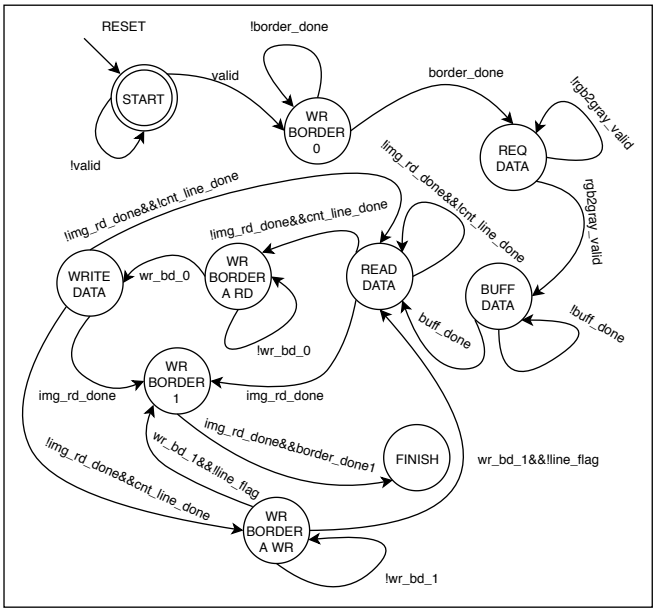


Figura 5.10: FSM - edge detect ctrl.

5.1.7 EDGE DETECT

Este módulo é responsável por realizar o processo de detecção de bordas. Como apresentado na seção 4.3, o bloco apresenta 2 FIFO síncronas, que são usadas para armazenar todos os *pixels* necessários para a realização do processo de convolução com a máscara de coeficientes.

O sinal buffer0_full indica para o módulo EDGE DETECT CTRL quando um novo dado pode ser enviado para o processamento. Este sinal é baseado no controle interno das FIFO, que utilizam os sinais *empty* e *full*, para saber quando o *buffer* está vazio ou cheio, respectivamente. Após os cálculos para obtenção do novo valor de *pixel*, resultante do processo de convolução e soma dos produtos, este valor é comparado com um valor de limiar, que é recebido neste módulo como uma parâmetro de entrada. Na Figura 5.11 é mostrado o diagrama de blocos do módulo EDGE DETECT.

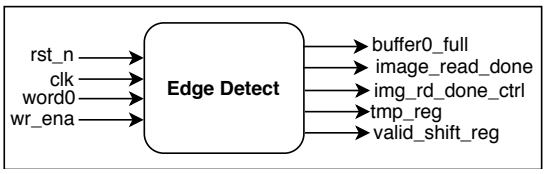


Figura 5.11: Módulo edge detect.

5.1.8 EDGE STORAGE

Este módulo é responsável por armazenar os *pixels* resultantes do processo de detecção de bordas. Para a realização deste processo, é necessário uma lógica de controle, para saber quando um *pixel* válido está disponível. Além disso, este módulo indica para qual espaço de endereçamento, os dados devem ser armazenados na memória SDRAM.

Outra função relevante deste módulo é indicar para o controle principal do sistema quando o processo de detecção de bordas foi finalizado. Na Figura 5.12 é mostrado o diagrama de blocos do módulo EDGE STORAGE.

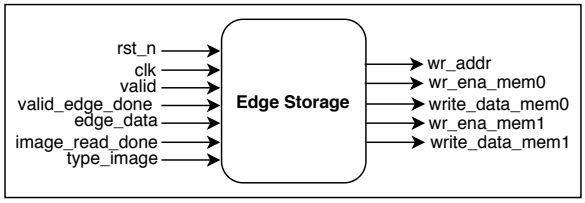


Figura 5.12: Módulo edge storage.

5.1.8.1 Lógica de Controle do módulo Edge Storage

Este módulo é composto por uma máquina de estados finita do tipo *Mealy* que possui 7 estados, Figura 5.13.

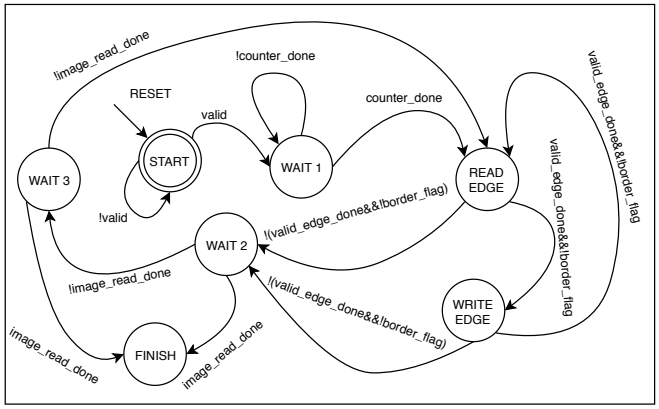


Figura 5.13: FSM - edge storage.

5.1.9 RGB2GRAY CTRL

Este módulo é responsável por obter os quadros de imagem da memória SDRAM, e enviar os dados para que seja realizada a operação de detecção de bordas. Para que isso seja possível, este módulo identifica de qual banco da memória SDRAM, os *pixels* são oriundos. Dessa forma, este módulo tem a capacidade de separar e organizar os 3 canais de cores (R , G, B), de forma que seja possível converter para o formato escala de cinza, pelo módulo CONV RGB2GRAY.

Este módulo tem a função de gerar o sinal que habilita a leitura na memória SDRAM, por este motivo ele deve ser capaz de identificar quando os dados provenientes da memória estarão disponíveis. Na Figura 5.14 é mostrado o diagrama de blocos do módulo RGB2GRAY CTRL.

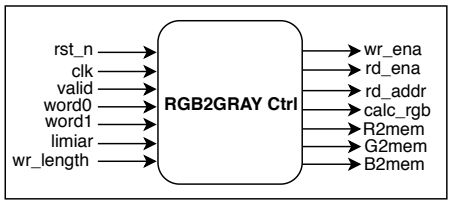


Figura 5.14: Módulo RGB2GRAY ctrl.

5.1.9.1 Lógica de Controle do módulo RGB2GRAY Ctrl

Este módulo é composto por uma máquina de estados finita do tipo *Mealy* que possui 9 estados, Figura 5.15.

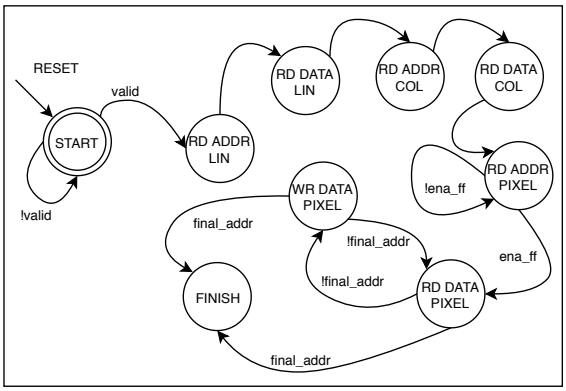


Figura 5.15: FSM - RGB2GRAY ctrl.

5.1.10 CONV RGB2GRAY

Este módulo é responsável por realizar a conversão dos *pixels* que estão no formato RGB, sendo que cada canal é representado por 8 bits, para o formato escala de cinza, representado por 8 bits.

Este bloco basicamente implementa a seguinte expressão:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \tag{5.1}$$

Na Figura 5.16 é mostrado o diagrama de blocos do módulo CONV RGB2GRAY.

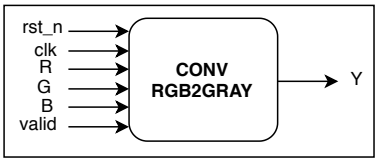


Figura 5.16: Módulo CONV RGB2GRAY.

5.1.11 COMPARE PIXEL CTRL

Este módulo é responsável por realizar o processo de detecção do objeto. Como apresentado na seção 4.4, o bloco apresenta uma lógica combinacional que permite a comparação entre os *pixels* de diferentes quadros de imagem, capturados em momentos distintos. Este módulo tem a característica de poder se comunicar com a memória SDRAM, tanto para leitura quanto para a escrita de dados. Desse modo, é possível obter os quadros de imagem armazenados nos dois bancos da memória SDRAM. Além disso, este módulo indica para o módulo FSM Ctrl quando um objeto foi detectado. Na Figura 5.17 é mostrado o diagrama de blocos do módulo COMPARE PIXEL CTRL.

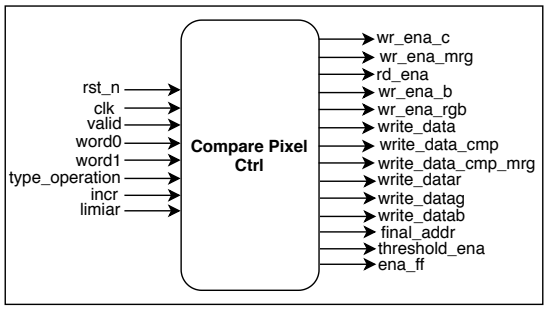


Figura 5.17: Módulo compare pixel ctrl.

5.1.11.1 Lógica de Controle do módulo Compare Pixel Ctrl

Este módulo é composto por uma máquina de estados finita do tipo *Mealy* que possui 6 estados, Figura 5.18.

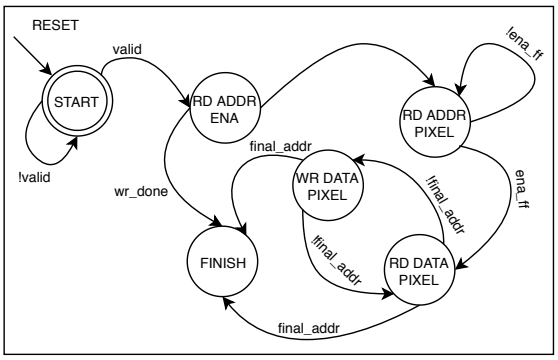


Figura 5.18: FSM - compare pixel ctrl.

5.1.12 VGA CTRL

Este módulo é responsável por fazer a integração com a interface VGA do dispositivo FPGA. As entradas deste módulo são as informações no formato RGB que serão lidas da memória SDRAM pelo módulo COMPARE PIXEL CTRL. As saídas deste módulo correspondem aos valores RGB a serem enviados para o VGA e os sinais de sincronização necessários. Na Figura 5.19 é mostrado o diagrama de blocos do módulo VGA Ctrl.

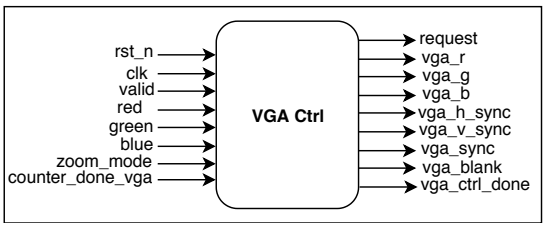


Figura 5.19: Módulo VGA ctrl.

5.1.13 Organização da Memória

As memórias SDRAM disponíveis no Intel DE2-115 FPGA foram usadas para armazenar quadros de vídeo capturados na resolução de $800x600$ ou $640x480$. A memória SDRAM consiste em 4 bancos (apenas 2 bancos são usados no projeto), cada banco

é uma matriz estruturada de linhas e colunas (8192 linhas x 512 colunas x 16 bits). O acesso para leitura da memória e gravação de dados na memória SDRAM é feito através de *bursts* pelo SDRAM Controller. A Figura 5.20 apresenta um mapa de memória com definições de espaço de endereço.

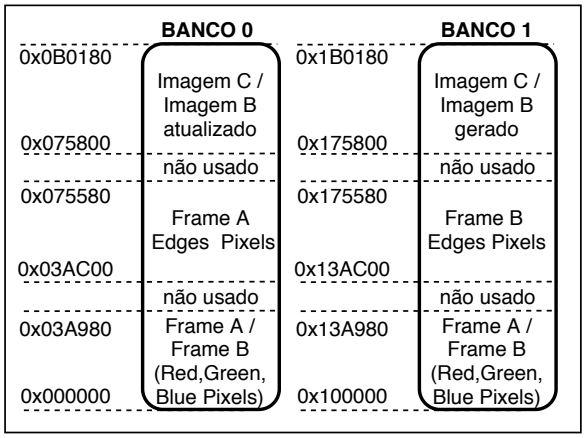


Figura 5.20: Organização da memória.

Como pode ser visualizado na Figura 5.20, a organização dos endereços da memória SDRAM está definida em três intervalos, em ambos os bancos de memória. Nos intervalos entre os endereços 0x000000 até 0x03A980 do banco 0, e os endereços 0x000000 até 0x13A980 do banco 1, estão os espaços de endereços destinados a todo quadro de imagem no formato RGB que foi capturado pela câmera. Para cada *pixel* RGB têm-se 24 bits de informação. Como cada endereço referente ao banco de memória é capaz de armazenar 16 bits, é necessária a utilização dos dois bancos de memória. No banco 0 de memória ficam armazenados os bits referentes às cores vermelha e verde do *pixel* RGB. No banco 1 ficam armazenados os bits referentes à cor azul do *pixel* RGB.

Nos intervalos entre os endereços 0x03AC00 até 0x075580 do banco 0, e os endereços 0x13AC00 até 0x175580 do banco 1, estão os espaços de endereços destinados a todo quadro de imagem que sofreu a operação de detecção de bordas. Após a operação de detecção de bordas, cada *pixel* do quadro de imagem é representado por 8 bits de informação. A partir disso, é possível armazenar dois quadros de imagens simultaneamente, nos dois bancos de memória.

Nos intervalos entre os endereços 0x075800 até 0x0B0180 do banco 0, e os endereços 0x175800 até 0x1B0180 do banco 1, estão os espaços de endereços destinados à imagem estática, resultante da operação de **Geração da imagem estática**, a

imagem auxiliar, resultante da operação de **Geração da imagem auxiliar**, e a imagem auxiliar atualizada, resultante da operação de **Atualização da imagem auxiliar**, apresentadas na seção 4.4. Cada *pixel* dos quadros de imagens, alocados nestes espaços de endereços, é representado por 8 bits de informação.

Como citado anteriormente, o sistema é capaz de armazenar e processar quadros de vídeo capturados na resolução de $800x600$. Para a resolução de $800x600$ têm-se 480.000 *pixels* de um quadro de imagem no formato RGB. Para atender os requisitos de tempo de processamento, foi necessário utilizar as duas memórias SDRAM para realizar acessos de leitura e escrita simultâneos dos *pixels* na memória. A organização da memória apresentada na Figura 5.20 é então aplicada para as duas memórias SDRAM disponíveis.

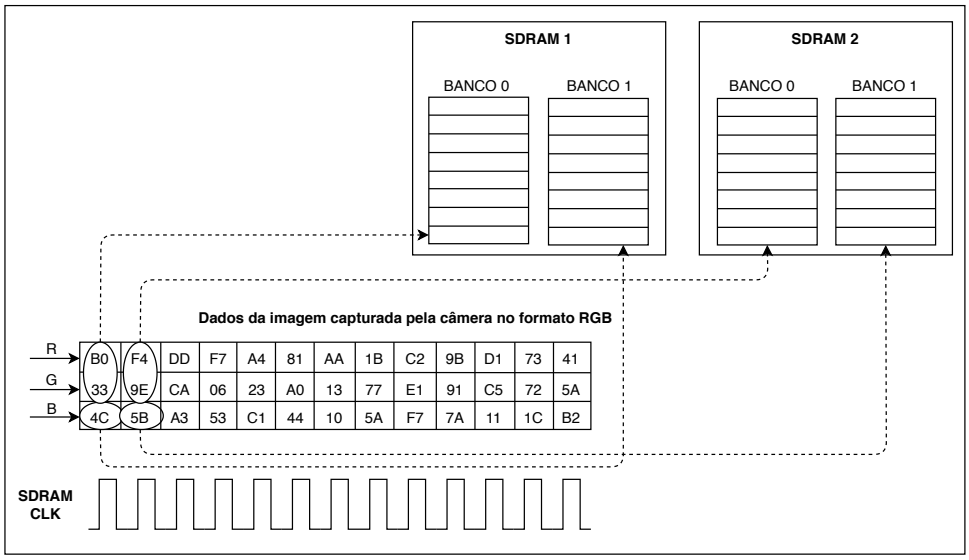


Figura 5.21: Armazenamento dos dados do quadro de imagem no formato RGB na memória.

Como pode ser visualizado na Figura 5.21, no canto inferior têm-se a linha do tempo (pulsos de *clock* do controlador da SDRAM) onde são apresentados os dados do quadro de imagem, capturados pela câmera. Os dados referem-se aos *pixels* RGB, representados por 24 bits de informação. No banco 0 da memória SDRAM 1 são armazenados os bits referentes às cores vermelha e verde do *pixel* RGB capturado no primeiro pulso de *clock*. No banco 0 da memória SDRAM 2 são armazenados os bits referentes às cores vermelha e verde do *pixel* RGB capturado no segundo pulso de *clock*. No banco 1 da memória SDRAM 1 são armazenados os bits referentes à cor

azul do *pixel* RGB capturado no primeiro pulso de *clock*. No banco 1 da memória SDRAM 2 são armazenados os bits referentes à cor azul do *pixel* RGB capturado no segundo pulso de *clock*.

O responsável por coordenar as escritas / leituras simultâneas, nas memórias SDRAM, é o controlador da SDRAM. A memória SDRAM e o controlador SDRAM operam na frequência de 100 MHz. No entanto, os módulos que recebem os *pixels*, dos quadros de imagem capturados pela câmera, operam na frequência de 50 MHz. Para superar incompatibilidades de frequência de *clock*, algumas FIFO são usadas como *buffers* de armazenamento de dados. O controlador SDRAM gera os sinais de controle, os endereços de memória e comandos para a memória SDRAM, a partir das informações recebidas de leitura e escrita nas FIFO. O controlador SDRAM facilita a comunicação com a memória, através do envio e recebimento de sinais para as FIFO.

Capítulo 6

Ambiente de Verificação Funcional Desenvolvido

A metodologia de verificação UVM foi aplicada neste projeto devido aos benefícios e recursos oferecidos, ambos apresentados na seção 3.2.5.1. Para a realização da verificação funcional, foi necessário o desenvolvimento de um ambiente estruturado, como citado na seção 3.2.5.3. O ambiente de verificação funcional deste projeto é composto por 5 componentes de verificação universal (UVC). A abordagem *bottom-up* foi aplicada no processo de desenvolvimento do ambiente de verificação funcional, ou seja, inicialmente os módulos que compõem o sistema foram verificados isoladamente. Após garantir que os módulos funcionam corretamente, é realizada uma integração entre todos os módulos e ambientes de verificação. O sistema foi modelado na linguagem dedicada Matlab com o objetivo de validar o comportamento das operações. Algumas operações descritas nas seções 6.1, 6.3 e 6.2 foram implementadas em linguagem C utilizando a representação de ponto fixo com o objetivo de proporcionar um modelo de referência para o ambiente de verificação.

6.1 Ambiente de Verificação do EDGE DETECT

O módulo EDGE DETECT é responsável por realizar o processo de detecção de bordas. Portanto, uma das principais funções do ambiente de verificação do EDGE DETECT é garantir que a operação de detecção de bordas seja efetuada corretamente. Para isso, o ambiente de verificação é composto de um Modelo de Referência,

validado e implementado em linguagem C. Os estímulos de entrada são gerados aleatoriamente, utilizando a técnica *Constrained Random Generation*, e enviados pelo componente *Driver*, tanto para o projeto em verificação ou DUV, quanto para o Modelo de Referência, como pode ser visualizado na Figura 6.1.

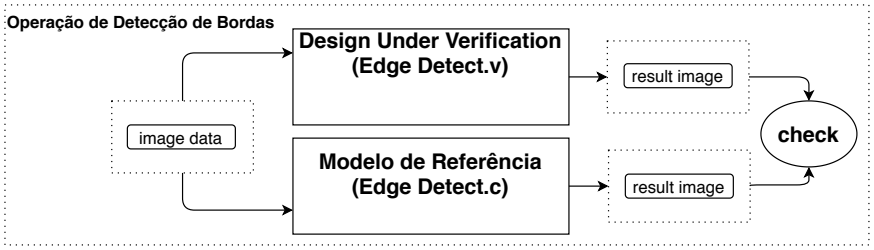


Figura 6.1: Modelo de referência do ambiente de verificação do edge detect

Um dos objetivos deste ambiente de verificação é garantir que a operação de detecção de bordas seja efetuada corretamente. O ambiente de verificação é composto por uma interface virtual, onde ficam as *Assertions*, utilizadas para garantir que a implementação dos sinais de entrada / saída do módulo EDGE DETECT tenham um comportamento correspondente ao definido na especificação do projeto de circuito digital. Além disso, os componentes *Driver* / *Monitor* promovem uma verificação dos protocolos de comunicação do módulo. Na Figura 6.2, pode ser visualizado o ambiente de verificação do módulo EDGE DETECT.

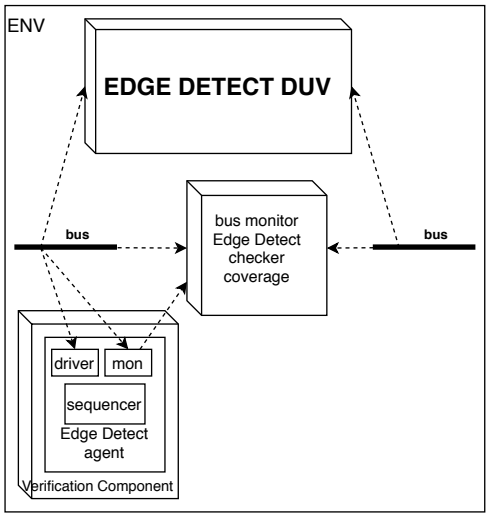


Figura 6.2: Ambiente de verificação do edge detect

6.2 Ambiente de Verificação do Compare Pixel CTRL

Este módulo é responsável por realizar o processo de detecção do objeto. Para realizar esta função, o módulo é composto por 3 principais operações: a geração da imagem estática, a geração da imagem auxiliar e a atualização da imagem auxiliar, como foi apresentado na seção 4.4. Portanto, uma das principais funções do ambiente de verificação do Compare Pixel CTRL, é garantir que as 3 operações citadas anteriormente, sejam efetuadas corretamente. Para isso, o ambiente de verificação é composto de um Modelo de Referência, validado e implementado em linguagem C. Os estímulos de entrada são gerados randomicamente, e enviados pelo componente *Driver*, tanto para o DUV, quanto para o Modelo de Referência, como pode ser visualizado na Figura 6.3.

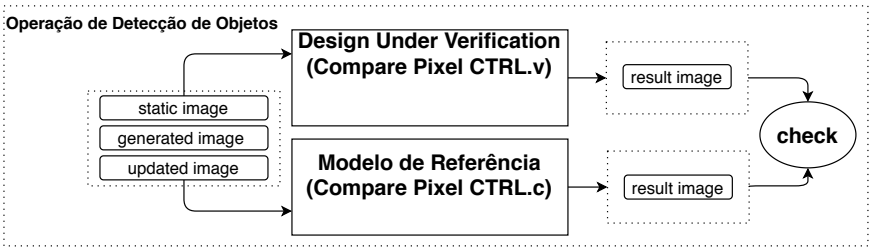


Figura 6.3: Modelo de referência do ambiente de verificação do compare pixel ctrl

O módulo Compare Pixel CTRL também possui um ambiente de verificação composto por uma interface virtual e os componentes *Driver* / *Monitor*. Na Figura 6.4, pode ser visualizado o ambiente de verificação do módulo Compare Pixel CTRL.

6.3 Ambiente de Verificação do CONV RGB2GRAY

Este módulo é responsável por realizar a conversão dos *pixels*, que estão no formato RGB, para o formato escala de cinza. Portanto, uma das principais funções do ambiente de verificação do CONV RGB2GRAY é garantir que a operação de conversão do formato RGB, para o formato escala de cinza, seja efetuada corretamente. Para isso, o ambiente de verificação é composto de um Modelo de Referência, validado e implementado em linguagem C. Os estímulos de entrada são gerados aleatoriamente,

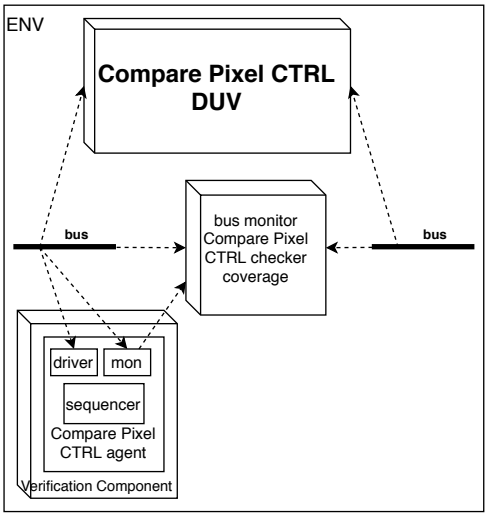


Figura 6.4: Ambiente de verificação do compare pixel ctrl

e enviados pelo componente *Driver*, tanto para o DUV, quanto para o Modelo de Referência, como pode ser visualizado na Figura 6.5.

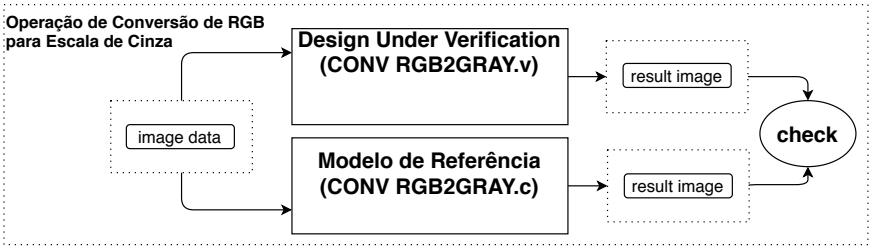


Figura 6.5: Modelo de referência do ambiente de verificação do CONV RGB2GRAY

O ambiente de verificação do CONV RGB2GRAY é composto por uma interface virtual e pelos componentes *Driver* / *Monitor*. Na Figura 6.6, pode ser visualizado o ambiente de verificação do módulo CONV RGB2GRAY.

6.4 Ambientes de Verificação dos Módulos de Controle de Leitura / Escrita na Memória

Os módulos Input Handler CTRL, RGB2GRAY CTRL, Edge Detect CTRL, Edge Storage e Compare Pixel CTRL têm, em comum, as funções de controle de escrita e leitura na Memória. Portanto, uma das principais funções dos ambientes de verificação dos Módulos de Controle de Leitura / Escrita na Memória é garantir que a leitura dos quadros de imagem, e a escrita na memória dos quadros de imagem

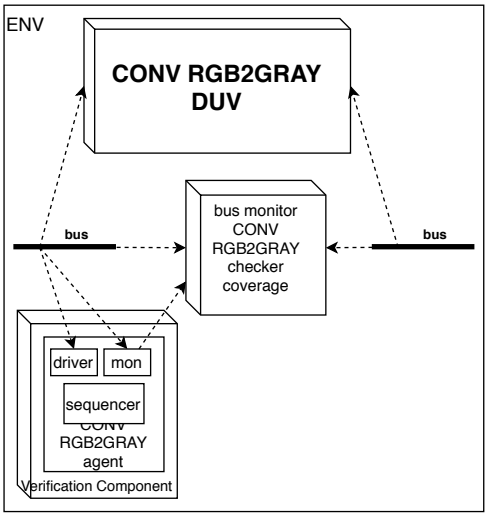


Figura 6.6: Ambiente de verificação do CONV RGB2GRAY

processados, sejam efetuadas corretamente. Desse modo, os ambientes de verificação são compostos de modelos de referência implementados no componente *Scoreboard* em System Verilog e UVM, que basicamente checam se os dados, que foram escritos em determinados endereços na memória, são iguais aos dados lidos dos mesmos endereços na memória.

Os ambientes de verificação dos módulos de controle de leitura / escrita na memória são compostos por interfaces virtuais e pelos componentes *Driver* / *Monitor*. Na Figura 6.7, pode ser visualizado um dos ambientes de verificação dos módulos de controle de leitura / escrita na memória.

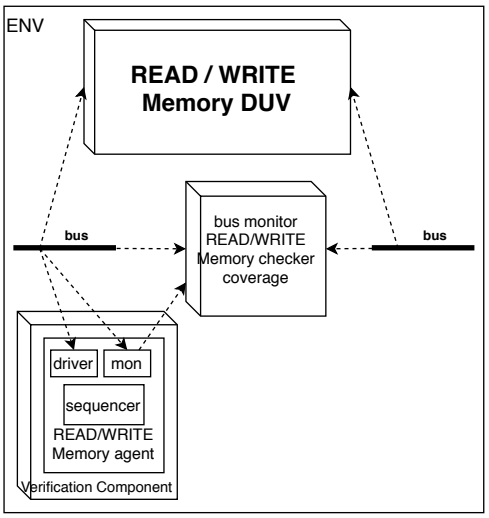


Figura 6.7: Ambiente de verificação dos módulos de controle de leitura / escrita na memória

6.5 Ambiente de Verificação do Topo do Projeto

A integração de todos os módulos desenvolvidos tem como função prover um sistema para a detecção de objetos. Portanto, uma das principais funções do ambiente de verificação do topo do projeto, é garantir que o sistema seja capaz de receber quadros de imagem de vídeo, e realizar a detecção de objetos na imagem. Desse modo, o ambiente de verificação é composto de um Modelo de Referência, implementado no componente *Scoreboard* em System Verilog e UVM, que realiza a comparação entre o quadro de imagem processado, obtido pelo DUV e o quadro de imagem processado, obtido pelo Modelo de Referência, implementado no software Matlab.

Além disso, o ambiente de verificação do topo reusa todos os ambientes de verificação que foram citados anteriormente. No entanto, o componente referente ao *Driver* dos ambientes reusados é desabilitado, apenas o componente Monitor continua realizando sua função, junto com o componente *Scoreboard*. A Figura 6.8, apresenta o ambiente de verificação do topo do projeto.

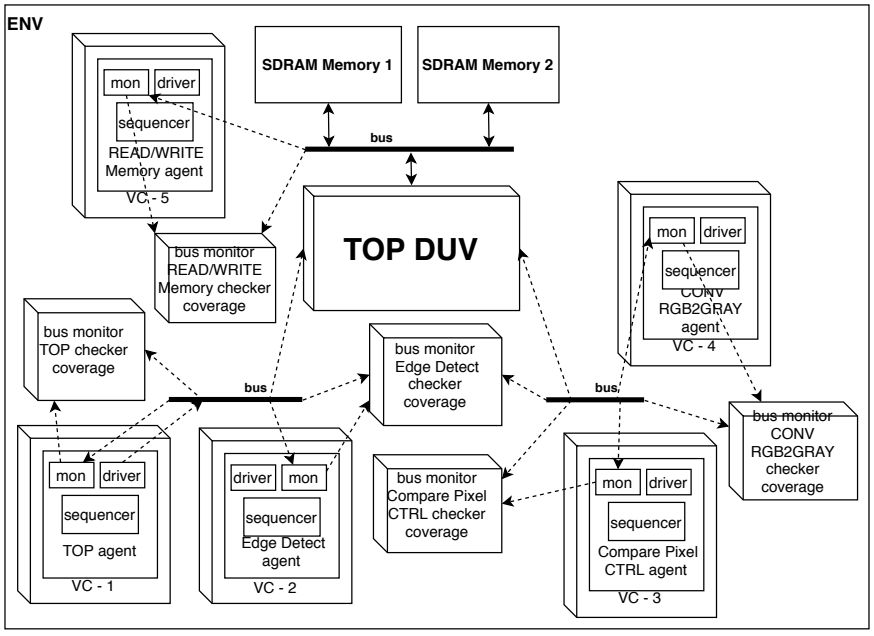


Figura 6.8: Ambiente de verificação do topo do projeto

6.6 *Assertions*

Para um projeto de circuitos digitais, que é composto por uma grande quantidade de módulos integrados, é fundamental a implementação das *Assertions*. Para cada

ambiente de verificação testado de forma isolada, as *assertions* são definidas no componente responsável por mapear a interface do módulo, especificando como deve ser o comportamento dos sinais de entrada e saída do bloco. Desse modo, é possível identificar de maneira otimizada a origem de um comportamento inesperado, de um sinal de entrada / saída de um sub-módulo do sistema. Nesse contexto, foram implementadas 35 *assertions*, incluindo todos módulos desenvolvidos, para garantir o correto funcionamento do sistema.

Capítulo 7

Resultados

Nesta seção, são apresentados todos os resultados obtidos, através da aplicação do fluxo de projeto em FPGA, da arquitetura de hardware para a detecção de objetos em tempo real. Inicialmente é apresentada uma análise sobre as restrições de tempo real definidas para este projeto. Além disso, são apresentados os resultados referentes à verificação funcional do projeto. Neste contexto, são apresentados os resultados obtidos através do desenvolvimento dos ambientes de verificação, bem como os resultados de cobertura funcional. Em seguida, são apresentados os resultados obtidos através da simulação funcional, onde o comportamento do sistema é simulado através do uso da ferramenta Incisive Simulator da Cadence Design Systems. Após a simulação funcional, foram obtidos os resultados referentes à síntese lógica do projeto de circuito digital, utilizando a ferramenta a Intel Quartus Prime.

Em seguida, são apresentados os resultados de desempenho, comparando o desempenho obtido, entre a implementação da operação de detecção de bordas, tanto em hardware, quanto em software. Por fim, são apresentados os resultados obtidos referentes à prototipação em FPGA do sistema desenvolvido. Como já foi citado anteriormente, para a execução dos testes, foi utilizada a câmera TRDB-D5M Terasic CMOS sensor, o kit FPGA Intel DE2-115, e um monitor VGA para exibição dos resultados.

7.1 Restrições de Tempo Real

Como foi visto na seção 2.4, o sistema de tempo real deve ter como base algumas restrições temporais previamente especificadas, de acordo com a aplicação a ser implementada. No caso deste projeto, as restrições de tempo estão voltadas para o período entre a aquisição do dado de entrada e a geração da saída adequada. A velocidade de processamento deve ser compatível com o fluxo de dados de entrada e saída.

O principal objetivo foi ter o máximo de tempo livre para satisfazer as restrições de tempo real em processamento de vídeo. Como foi visto na seção 5.1.13, a utilização das duas memórias SDRAM disponíveis, o controlador SDRAM e os *buffers* de entrada e saída de dados, permitiram a transferência de 32 bits a cada pulso de *clock*. Considerando a transferência de um quadro de imagem com a resolução $640x480$, foi necessário a utilização de 76800 ciclos de *clock*. Portanto, analisando o tempo para aquisição e transferência do quadro de imagem, foi observado o tempo de 3,072 ms.

Baseado no padrão VGA definido em [33], a Figura 7.1 ilustra os requisitos de tempo vertical para cada quadro que é exibido em um monitor VGA, considerando um monitor com atualização de 60 Hz no modo $640x480$ e um *pixel clock* de 25 MHz.

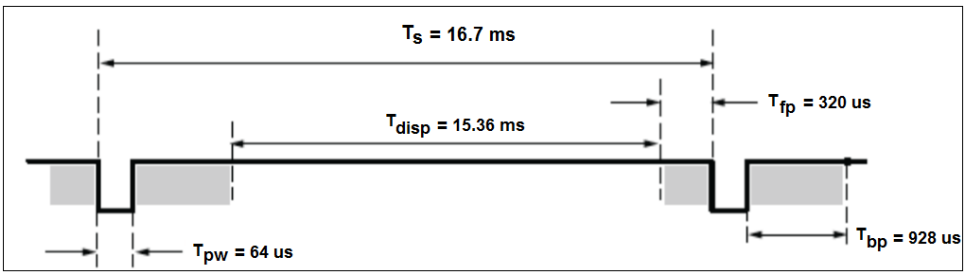


Figura 7.1: Tempo de controle VGA com atualização de 60 Hz e *pixel clock* de 25 MHz.

Na Figura 7.1, está representado o tempo necessário para exibição de uma imagem no monitor VGA, que é o tempo de 16,7 ms. A partir do resultado obtido do tempo para aquisição e transferência do quadro de imagem, que foi de 3,072 ms, é possível afirmar que cerca de 13 ms estão disponíveis para processamento de tempo real. O processamento para detecção de bordas e identificação do objeto para um

quadro de imagem com a resolução $640x480$ foi de aproximadamente 6,17 ms. Levando em consideração que o tempo para aquisição e processamento do quadro de imagem é menor que os 16,7 ms especificados no padrão VGA definido em [33], é possível afirmar que a restrição de tempo foi atendida.

7.1.1 Resultados da Cobertura Funcional e Cobertura de Código

Para medir o progresso da verificação, é realizada a cobertura da verificação, ou seja, é necessária a definição da extensão da verificação problema. O objetivo de medir a cobertura funcional é analisar o progresso da verificação, a partir da perspectiva dos requisitos funcionais do projeto de circuito digital. Os requisitos funcionais são descritos na especificação do projeto de circuito digital, como foi apresentado na seção 3.2.1. Por exemplo, o sistema tem um requisito funcional que determina a identificação dos objetos, através do processo de detecção de borda utilizando o operador de Sobel. A partir disso, definiu-se que o módulo Edge Detect tem um requisito funcional, que é realizar a operação de detecção de borda, utilizando o operador de Sobel corretamente. Portanto, a cobertura funcional deve definir no plano de verificação, para todos os módulos do projeto de circuito digital, uma lista de recursos a serem testados, que determinará a qualidade da verificação. A Tabela 7.1 mostra os resultados da cobertura funcional dos módulos do projeto.

Tabela 7.1: Resultados da cobertura funcional

Módulo	*Status*
Edge Detect	100%
Compare Pixel CTRL	100%
CONV RGB2GRAY	100%
Input Handler CTRL	100%
RGB2GRAY CTRL	100%
Edge Detect CTRL	100%
Edge Storage	100%
Topo do Projeto	100%

A cobertura do código mede a execução do código RTL. A coleta de informações de cobertura de código, incluindo cobertura de *statement*, *branch*, *block*, *toggle*, cobertura da FSM, é amplamente automática. A cobertura do código pode ser considerada como uma medida quantitativa da execução do código do DUV. Por exemplo, se um módulo RTL tem 10 linhas de código, e apenas 8 linhas de código foram executadas, então este módulo RTL apresentará um cobertura de *statement* de 80%. A cobertura de *block* considera blocos ramificados de *if/else*, *case*, *wait*, *while*, *for*, etc. A cobertura de *branch* reporta a condição verdadeira ou falsa das condições, como *if-else* ou *case*. A cobertura de *toggle* reporta qual sinal não mudou de estado, durante toda a execução. A cobertura da FSM reporta quantas vezes um estado foi visitado, transitado e quantas sequências foram cobertas, numa máquina de estados finita. A Tabela 7.2 mostra os resultados da cobertura de código dos módulos do projeto.

Tabela 7.2: Resultados da cobertura de código

Módulo	*Status*
Edge Detect	100%
Compare Pixel CTRL	100%
CONV RGB2GRAY	100%
Input Handler CTRL	100%
RGB2GRAY CTRL	100%
Edge Detect CTRL	100%
Edge Storage	100%
Topo do Projeto	100%

7.2 Resultados da Simulação Funcional

A validação do funcionamento do código RTL foi realizada através do método de simulação, utilizando a ferramenta de verificação Incisive Simulator da Cadence Design Systems. Como citado na seção **??**, na verificação funcional dos módulos realizada de forma isolada, os estímulos de entrada foram gerados a partir da técnica

Constrained Random Generation, ou seja, a geração aleatória de estímulos de entrada, que obedecem a um conjunto de restrições de entrada. No entanto, para a simulação funcional do topo do projeto, foram utilizados os testes direcionados, onde foram escolhidas algumas imagens (cenários de testes) predefinidas, utilizadas como estímulos de entrada.

Para o caso de teste direcionado, quando o sistema é iniciado, apenas um objeto encontra-se estático no cenário de teste, como pode ser visto na imagem à esquerda na Figura 7.2. Esta imagem pode ser considerada como o primeiro quadro de imagem de vídeo capturado. Após a captura do primeiro quadro de imagem, ocorre o processo de detecção de borda. A imagem obtida é armazenada na memória SDRAM e exibida no monitor VGA, como pode ser visto na imagem à direita na Figura 7.2.

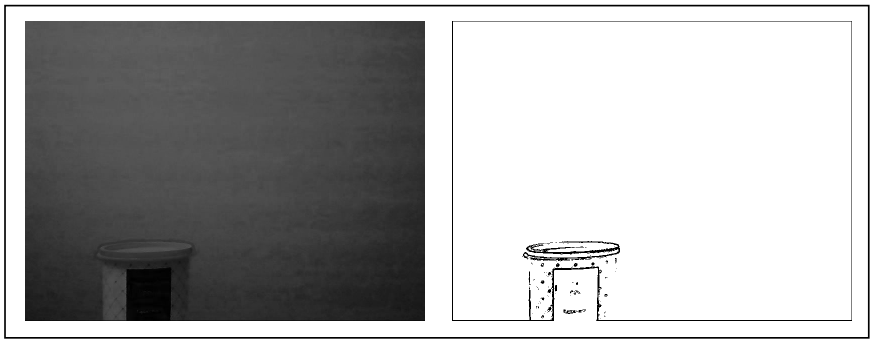


Figura 7.2: 1° quadro de imagem de vídeo capturado

O próximo passo realizado pelo algoritmo do sistema é aguardar um tempo determinado para capturar um novo quadro de imagem de vídeo, então o sistema aguarda 1 segundo e captura um novo quadro. Como pode ser visualizado na imagem à esquerda da Figura 7.3, percebe-se que não ocorreu uma mudança, com relação ao primeiro quadro de imagem de vídeo capturado. A partir do 1° e do 2° quadro obtidos anteriormente, o sistema considera que o cenário estático é o quadro de imagem de vídeo apresentado na imagem à direita na Figura 7.3.

Após a obtenção do cenário estático, o sistema novamente aguarda um período de tempo determinado, para capturar um novo quadro de imagem de vídeo, então o sistema aguarda 1 segundo e captura o novo quadro. Como pode ser visto na imagem à esquerda na Figura 7.4, percebe-se que ocorreu a inserção de um novo objeto na imagem. Na imagem à direita na Figura 7.4, têm-se o resultado da detecção de bordas, do 3° quadro de imagem de vídeo capturado.

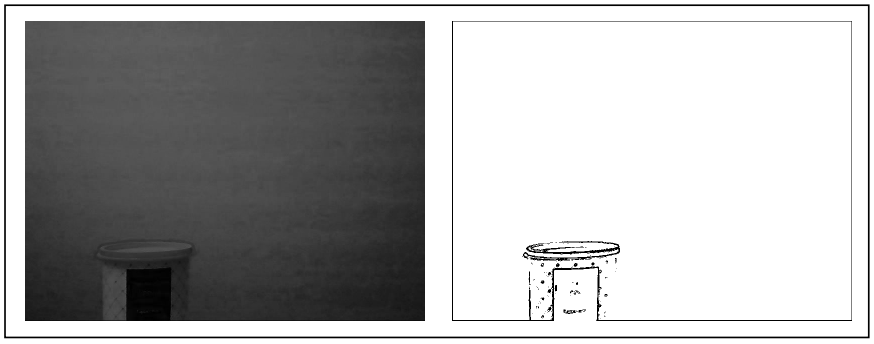


Figura 7.3: 2° quadro de imagem de vídeo capturado

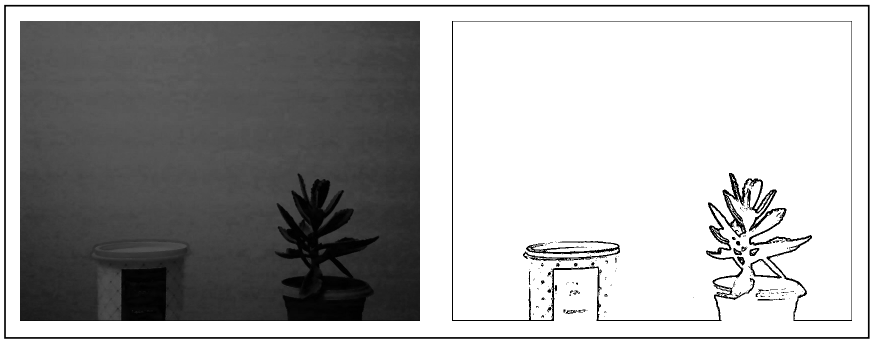


Figura 7.4: 3° quadro de imagem de vídeo capturado

A partir deste ponto, o sistema captura novos quadros de imagem de vídeo, e segue comparando em relação ao cenário estático, até que o limiar estabelecido seja atingido. Na imagem à esquerda na Figura 7.5, têm-se o 7° quadro de imagem de vídeo obtido. A imagem à direita na Figura 7.5, apresenta o resultado armazenado na memória SDRAM do processamento de identificação do objeto em uma cena fixa, quando o processo atinge o limiar estabelecido. Ou seja, um novo objeto foi identificado em relação ao cenário estático. É possível perceber que os valores dos *pixels* do objeto identificado são diferentes do objeto do cenário estático.

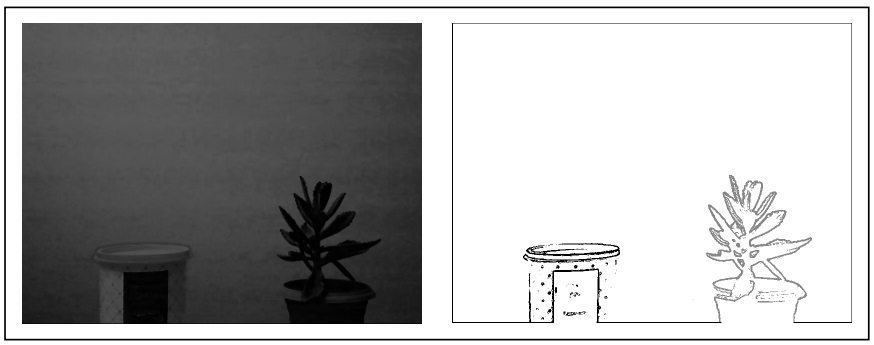


Figura 7.5: 7° quadro de imagem de vídeo capturado

Após o sistema detectar que o limiar foi atingido, é realizado uma junção entre o quadro de imagem de vídeo armazenado na memória, que identifica o objeto, e o último quadro de imagem de vídeo capturado. O resultado é apresentado na Figura 7.6.

Figura 7.6: Resultado da detecção do objeto em uma cena fixa

7.3 Resultados da Síntese Lógica

Após a validação funcional em simulação, foi realizada a síntese lógica do RTL na ferramenta Intel Quartus Prime para o FPGA da família Cyclone IV E, com o identificador do dispositivo EP4CE115F29C7. A Tabela 7.3, apresenta os resultados obtidos na etapa de síntese lógica.

Tabela 7.3: Resultado da síntese para FPGA

Elementos	Utilização
Total de elementos lógicos	3,884 / 114,480 (3%)
Total de registradores	2021
Total de pinos	428 / 529 (81%)
Total de bits de memória	98,360 / 3,981,312 (2%)
Multiplicadores embarcados 9-bit elementos	2 / 532 (<1%)
Total PLLs	1 / 4 (25%)

Como é possível observar na Tabela 7.3, a arquitetura de hardware desenvolvida e implementada obteve uma baixa utilização dos recursos disponíveis do FPGA. Dessa forma torna-se possível a implementação do sistema em FPGA de baixo custo.

7.4 Resultados de Desempenho

A partir da implementação de uma arquitetura de hardware, bem como a implementação de um sistema no software Matlab, para a detecção de objetos, foi possível realizar uma análise referente ao desempenho, considerando o tempo consumido para efetuar a operação de detecção de objetos.

O software Matlab foi executado a partir de um computador com um processador Intel® Core™ i5-5200U e 8GB de memória RAM.

Na Figura 7.7, é apresentado um gráfico gerado, a partir das medições do tempo de execução para cada caso, hardware e software.

Para realizar a medição no software Matlab foi utilizada a função **Profiler**, para coletar o tempo de execução que a operação de detecção de objetos consome. Para realizar a medição no hardware RTL foi utilizado a simulação a nível de portas (*Gate Level Simulation*), na ferramenta Incisive Simulator da Cadence Design Systems. A partir dos resultados do gráfico, é possível afirmar que houve *speedup* de 310x na execução da operação de detecção de objetos, com a implementação do código em RTL. É possível perceber um aumento de desempenho considerável da execução em FPGA, quando comparado à implementação em software Matlab.

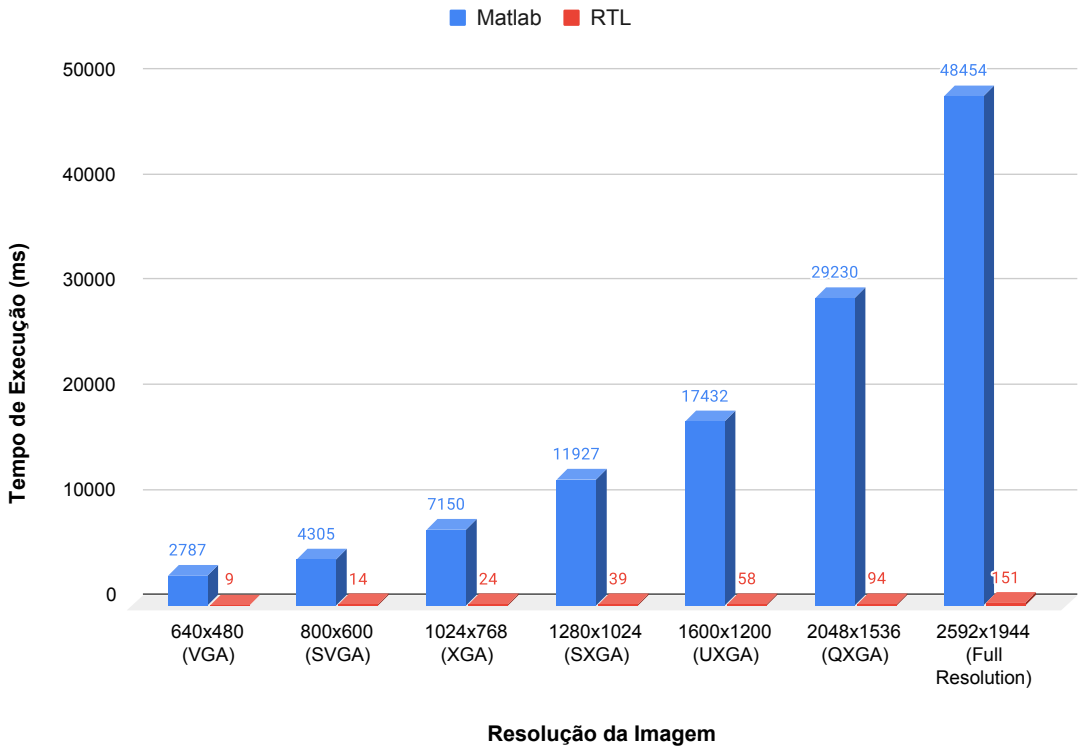


Figura 7.7: Resultados de desempenho

7.5 Resultados da Prototipação em FPGA

Para realizar a prototipação em FPGA, foi desenvolvido um cenário de teste similar ao apresentado na seção 7.2, a diferença é que para este cenário de teste, são utilizados 3 objetos. Quando o sistema é iniciado, apenas um objeto encontra-se estático no cenário de teste, como pode ser visto na Figura 7.8. Esta imagem pode ser considerada como o primeiro quadro de imagem de vídeo capturado. Após a captura do primeiro quadro de imagem, ocorre o processo de detecção de borda.

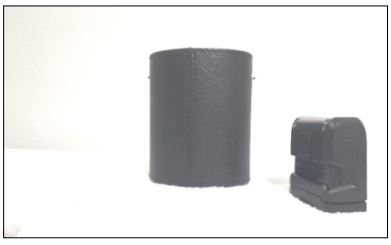


Figura 7.8: 1° quadro de imagem de vídeo capturado (FPGA)

O próximo passo realizado pelo algoritmo do sistema é aguardar um tempo determinado para capturar um novo quadro de imagem de vídeo, então o sistema

aguarda 1 segundo e captura um novo quadro. O novo quadro de imagem conterá os mesmos objetos capturados no primeiro quadro de imagem de vídeo. Com isso, a imagem estática armazenada na memória será o resultado da detecção de borda na imagem inicial da Figura 7.8.

Após a obtenção do cenário estático, o sistema novamente aguarda um período de tempo determinado para capturar um novo quadro de imagem de vídeo, então o sistema aguarda 1 segundo e captura o novo quadro. Como pode ser visto na Figura 7.9, percebe-se que ocorreu a inserção de um novo objeto na imagem.

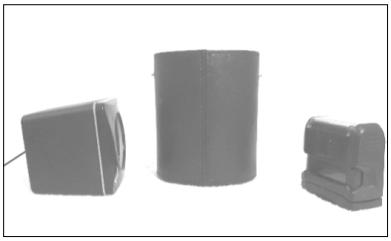


Figura 7.9: 3° quadro de imagem de vídeo capturado (FPGA)

A partir deste ponto, o sistema captura novos quadros de imagem de vídeo e segue comparando em relação ao cenário estático, até que o limiar estabelecido seja atingido.

Após o sistema detectar que o limiar foi atingido, é realizado uma junção entre o quadro de imagem de vídeo armazenado na memória, que identifica o objeto, e o último quadro de imagem de vídeo capturado. O resultado é apresentado na Figura 7.10.

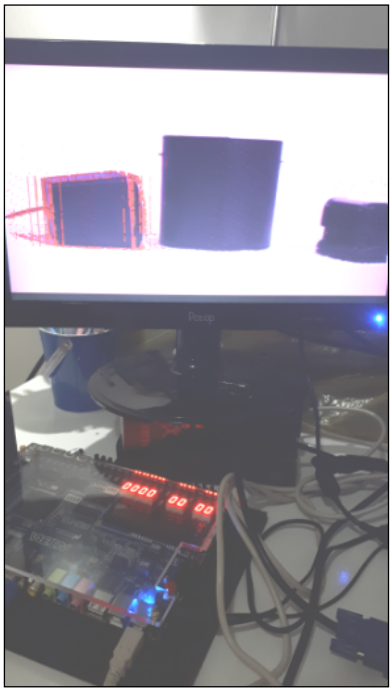


Figura 7.10: Resultado da detecção do objeto em uma cena fixa (FPGA)

Capítulo 8

Conclusões

Este trabalho apresentou o desenvolvimento de uma arquitetura de hardware, para a detecção de objetos em tempo real e a prototipação da solução desenvolvida em FPGA. A partir do estudo dos conceitos e técnicas de processamento digital de imagens a operação de detecção de bordas foi implementada utilizado o operador de Sobel.

A metodologia aplicada foi a execução das principais etapas do fluxo de projeto em FPGA, de forma ampla, considerando os recursos e técnicas mais utilizados atualmente. A partir da correta aplicação de todas as etapas do fluxo de projeto em FPGA, observou-se que o propósito do trabalho foi alcançado. A aplicação foi desenvolvida considerando todas as especificações funcionais. A modelagem do sistema foi realizada através da implementação no software Matlab e a verificação funcional realizada validou o correto funcionamento dos módulos implementados em código RTL. A prototipação em FPGA constatou o funcionamento e a integração de todo o sistema proposto.

Através da utilização da câmera TRDB-D5M Terasic CMOS sensor, integrada ao kit FPGA DE2-115 e o monitor VGA, foi possível realizar diferentes cenários de teste, que comprovaram a eficiência do algoritmo de detecção de objetos em uma cena fixa em vídeos digitais. A partir dos testes foram obtidos resultados com um *speedup* de 310x para a implementação do algoritmo de detecção de objetos em FPGA, quando comparada à implementação no software Matlab. Além disso, os resultados de síntese lógica mostraram que a arquitetura de hardware implementada teve uma baixa utilização dos recursos do FPGA, possibilitando que o sistema seja

implementado em um dispositivo de baixo custo. Em adição, com a disponibilidade de recursos do FPGA é possível aumentar o desempenho através da otimização do código RTL.

Uma das principais características da arquitetura desenvolvida é a reusabilidade que os módulos de processamento de imagens fornecem. Desta forma, alguns módulos poderão ser usados em outros projetos de hardware digital, com ênfase em visão computacional, com algumas modificações.

Como melhoria para este trabalho, é almejado a implementação de um filtro digital que possa eliminar ruídos que foram identificados durante o processo como, por exemplo, a interferência da luminosidade capturada pela câmera, nos cenários de teste. Para isso, é necessário a inserção desse requisito no documento de especificação funcional e, como consequência, a atualização da microarquitetura, num processo de realimentação do fluxo de projeto em FPGA. Outro trabalho futuro seria a integração do sistema com outros protocolos de comunicação, visando a ampliação de funcionalidades do mesmo.

Referências Bibliográficas

[1] GUENNOUNI, S., AHAITOUF, A., MANSOURI, A., "Multiple object detection using OpenCV on an embedded platform". In: *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*, pp. 374–377, 2014.

[2] GOYAL, K., AGARWAL, K., KUMAR, R., "Face detection and tracking: Using OpenCV". In: *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, v. 1, pp. 474–478, 2017.

[3] YOGITHA, S., SAKTHIVEL, P., "A distributed computer machine vision system for automated inspection and grading of fruits". In: *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*, pp. 1–4, 2014.

[4] MONTEIRO, A. C. B., OTHERS, "Proposta de uma metodologia de segmentação de imagens para detecção e contagem de hemácias e leucócitos através do algoritmo WT-MO", 2019.

[5] PALEKAR, R. R., PARAB, S. U., PARIKH, D. P., et al., "Real time license plate detection using openCV and tesseract". In: *2017 International Conference on Communication and Signal Processing (ICCSP)*, pp. 2111–2115, 2017.

[6] ALCANTARA, G. K. L., EVANGELISTA, I. D. J., MALINAO, J. V. B., et al., "Head Detection and Tracking Using OpenCV". In: *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pp. 1–5, 2018.

[7] CHANDAN, G., JAIN, A., JAIN, H., et al., "Real Time object detection and Tracking using Deep Learning and openCV". In: *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 1305–1308, 2018.

[8] MOORE, A., WILSON, R., "FPGA for dummies", 2017.

[9] SUNDARARAJAN, P., "High performance computing using FPGAs", *Xilinx white paper: FPGAs*, pp. 1–15, 2010.

[10] GONZALEZ, R. C., WOODS, R. E., *Digital Image Processing*. 4th ed. Pearson, 2017.

[11] MARQUES, O., *Practical Image And Video Processing Using MATLAB*. 1st ed. John Wiley Professio, 2011.

[12] BOVIK, A., *Handbook of Image and Video Processing*. 2nd ed. Elsevier, 2005.

[13] ANNADURAI, S., SHANMUGALAKSHMI, R., *Fundamentals of Digital Image Processing*. 1st ed. Pearson, 2006.

[14] ZHANG, Y., *Image Processing*. 2nd ed. De Gruyter, 2017.

[15] MCANDREW, A., *A Computational Introduction to Digital Image Processing*. 2nd ed. Chapman and Hall/CRC, 2015.

[16] FARINES, J.-M., FRAGA, J. D. S., OLIVEIRA, R. D., "Sistemas de tempo real", *Escola de Computação*, v. 2000, pp. 201, 2000.

[17] CHAO LI, SOULEYMANE BALLA-ARABE, F. Y., *Architecture-Aware Optimization Strategies in Real-time Image Processing*. 1st ed. Wiley-ISTE, 2017.

[18] RISHA, K., KUMAR, A. C., "Novel method of detecting moving object in video", *Procedia Technology*, v. 24, n. 1, pp. 1055–1060, 2016.

[19] SINGLA, N., "Motion detection based on frame difference method", *International Journal of Information & Computation Technology*, v. 4, n. 15, pp. 1559–1565, 2014.

[20] LU, N., WANG, J., WU, Q., et al., "An Improved Motion Detection Method for Real-Time Surveillance." *IAENG International Journal of Computer Science*, v. 35, n. 1, 2008.

[21] LLORENTE, C. A., DADIOS, E. J. P., MONZON, J. A. B., et al., "FPGA-based Object Detection and Classification of an Image", *International Journal of Engineering Technology*, v. 7, n. 4.16, pp. 83–86, 2018.

[22] JADHAV, S., NARVEKAR, R., MANDAWALE, A., et al., "FPGA Based Object Tracking System". In: *2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 826–829, April 2015.

[23] PAWASKAR, M. C., NARKHEDE, N., ATHALYE, S. S., "Moving Object Detection Using FPGA", *International Journal of Emerging Trends and Technology in Computer Science*, v. 3, n. 3, pp. 219–222, 2014.

[24] HANCHINAMANI, S. R., SARKAR, S., BHAIRANNAWAR, S. S., "Design and implementation of high speed background subtraction algorithm for moving object detection", *Procedia computer science*, v. 93, pp. 367–374, 2016.

[25] CHU, P. P., *FPGA prototyping by Verilog examples: Xilinx Spartan-3 version*. John Wiley & Sons, 2011.

[26] MONMASSON, E., CIRSTEA, M. N., "FPGA design methodology for industrial control systems—A review", *IEEE transactions on industrial electronics*, v. 54, n. 4, pp. 1824–1842, 2007.

[27] CHEN, D., CONG, J., PAN, P., et al., "FPGA design automation: A survey", *Foundations and Trends® in Electronic Design Automation*, v. 1, n. 3, pp. 195–330, 2006.

[28] WILSON, P., *Design Recipes for FPGAs: Using Verilog and VHDL*. 2nd ed. Newnes, 2015.

[29] LORANDEL, J., PRÉVOTET, J.-C., HELARD, M., "Fast power and performance evaluation of FPGA-based wireless communication systems", *IEEE Access*, v. 4, pp. 2005–2018, 2016.

[30] WILE, B., GOSS, J., ROESNER, W., *Comprehensive Functional Verification: The Complete Industry Cycle*. Morgan Kaufmann, 2005.

[31] HEIGHT, H., *A practical guide to adopting the universal verification methodology (UVM)*. Lulu. com, 2010.

[32] TERASIC, "TRDB-D5M: Terasic D5M Hardware Specification", April 2008, Disponível em: https://www.terasic.com.tw/en/.

[33] ASSOCIATION, V. E. S., OTHERS, "VESA and industry standards and guidelines for computer display monitor timing (DMT)", 2007.

[34] BAILEY, D. G., *Design for Embedded Image Processing on FPGAs*. 1st ed. Wiley-IEEE Press, 2011.

[35] IEEE, *IEEE Standard Verilog Hardware Description Language*, April 2006, IEEE Std 1364-2005, Disponível em: https://standards.ieee.org/standard/1364-2005.html.

[36] RAFIQUZZAMAN, M., *Fundamentals of Digital Logic and Microcontrollers*. 6th ed. Wiley, 2014.

[37] DAVID A. PATTERSON, J. L. H., *Computer Organization and Design: The Hardware/Software Interface*. 5th ed. Morgan Kaufmann Publishers, 2013.

[38] ETHEREDGE, C., *Goslow: Design and implementation of a scalable camera array for high-speed imaging*, Master's Thesis, University of Twente, 2016.

[39] MEYER-BAESE, U., *Digital signal processing with field programmable gate arrays*. 3rd ed., v. 65. Springer Publishing Company, Incorporated, 2007.

[40] SADROZINSKI, H. F.-W., WU, J., *Applications of field-programmable gate arrays in scientific research*. CRC Press, 2016.

Apêndice A

Artigo apresentado no *Workshop on Circuits and System Design* - WCAS 2019

Implementation of a Real-Time Processing System for Object Detection in Digital Videos using FPGA

Lauê Rami S. C. Jesus*
Universidade Federal da Bahia
Salvador, Bahia, Brazil
lauerami@gmail.com

Wagner L. A. Oliveira
Universidade Federal da Bahia
Salvador, Brazil
wlao@uol.com.br

Paulo C. M. A. Farias
Universidade Federal da Bahia
Salvador, Brazil
pcma.farias@gmail.com

ABSTRACT

This paper discusses the implementation of a Real-Time Processing System for Object Detection in Digital Videos. The system consists of real-time image processing and this application is suitable for implementation in FPGA due to the algorithm features and the device resources. The proposed hardware architecture was designed and synthesized for Intel DE2-115 FPGA. The main concepts, techniques, the methodology used for development the design will be presented. The results are verified in real time with an input video from TRDB-D5M Terasic CMOS sensor, DE2-115 FPGA device and VGA monitor for displaying images.

CCS CONCEPTS

• **Hardware → Programmable logic elements**; *Combinational circuits*; *Finite state machines*; *Sequential circuits*; Simulation and emulation.

KEYWORDS

FPGA, Object detection, RTL Design, Edge detection

ACM Reference Format:
Lauê Rami S. C. Jesus, Wagner L. A. Oliveira, and Paulo C. M. A. Farias. 2019. Implementation of a Real-Time Processing System for Object Detection in Digital Videos using FPGA. In *Proceedings of Workshop on Circuits and System Design (WCAS2019).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Object detection is the act of finding the location of an object in an image. Today, object detection algorithms are used in applications such as surveillance [5] and security, tracking objects, face detection [4], industrial inspection [7], self driving cars, and others. Video surveillance aims to analyze video data frames to detect objects of interest and generate security alarms.

Computer Vision strategies for object detection are commonly developed in software. There are different works which study the implementation of different object detection algorithms in software. Image Processing Algorithm implemented in Matlab and OpenCV as presented in [6], [1], [2] could be improved in terms of performance and power efficiency. Many object detection techniques use edge detection operators as a step for processing image data. Edge detection methods consist of image analysis and structural information extraction. This extracted edge information reduces the amount of data to be processed. The edge detection process is computationally exhaustive because of the large number of operations, for each edge detected pixel value - about 20 logic and arithmetic operations are executed. Therefore, we target our object detection application onto an FPGA.

The reconfigurable FPGA provides large arrays of programmable logic resources. The FPGA device also offers several advantages, including the parallel processing of data, which provides the ability to implement complex digital functions and this features are suitable for the implementation of real time image processing algorithms.

In this context, this work presents a Real-Time Processing System for Object Detection in digital videos. Section 2 presents concepts related to FPGAs; Section 3 discusses image processing concepts that were employed in this project; Section 4 presents the design methodology; Section 5 presents results found in simulation environment and FPGA prototyping; Section 6 presents a conclusion for the paper.

2 DIGITAL IMAGE PROCESSING

2.1 Image Segmentation

Segmentation subdivides an image into regions or objects. The segmentation process must stop when the objects of interest in an application are isolated. Image segmentation algorithms are based on one of two basic properties of intensity values: discontinuity and similarity.

In the first category, the approach is to split an image based on abrupt changes of intensity, such as edges in an image. The second category provides the image partitioning in regions that are similar according to a set of predefined criteria [3].

2.2 Edge Detection

Edge detection is the most common approach to detecting significant discontinuities in the gray level. A border is a set of pixels connected and located at the border between two regions. The edges are regions of the image where a change of intensity occurs in a certain space interval, in a certain direction. This corresponds to high spatial derivative regions, which contain high spatial frequency.

An operator that is sensitive to abrupt variations in intensity will operate as an edge detector. A derivative operator does exactly this function. After obtaining the intensity values of the image and

WCAS2019, August 26–30, 2019, São Paulo, Brazil

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

calculating the points where the derivative is a maximum point, the marking of the edges is obtained [3].

In this work we used the pair of masks for the magnitude approximation of the gradient, called Sobel operators, presented in (1).

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (1)$$

3 METHODOLOGY

This work has employed digital image processing techniques and system methodologies for embedded hardware focusing FPGAs. The FPGA design flow adopted a top-down approach with well defined design levels.

3.1 System Modeling

The application was modeled with the dedicated Matlab language, through functions for the image acquisition process, the RGB format to grayscale conversion, the edge detection using the Sobel operator and the comparison of the pixels of different image frames for identification of fixed objects. Such functions were implemented in software aiming to support the functional verification and the system validation.

3.2 Edge Detection Algorithm

According to [3] the edge detection algorithm can be implemented by running the mask through the image as can be seen in Figure 1.

This procedure includes computing the sum of the products of the coefficients with the gray levels contained in the region covered by the mask. In this process the mask scans the entire image.

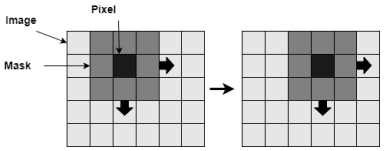


Figure 1: Convolving a mask with an image

The hardware architecture for edge detection algorithm implementation has a structure that includes two FIFOs (First In First Out) and shift registers. Each pixel can be represented by an image frame. At each clock cycle the pixels are shifted in the same direction, with the oldest one being removed. Each pixel stored in the FIFOs or in the shift registers represents one line of the image. The FIFOs size is defined by the number of columns in the image and the mask size used. Based on the architecture of Figure 2, it is possible to obtain the calculated pixels from multiplication and addition operations between the mask coefficients and the neighboring pixels in the region of interest.

3.3 Object Detection Algorithm

The application used in this work aims to implement an algorithm that identifies fixed objects from images captured by a camera.

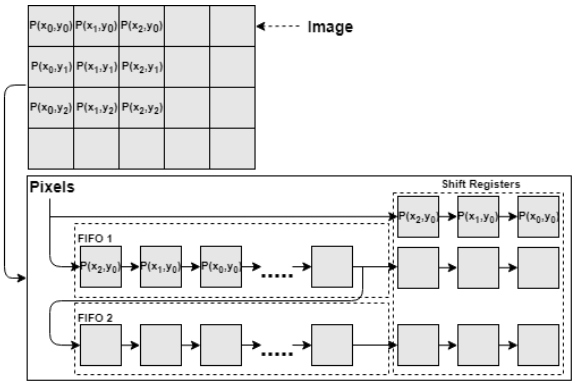


Figure 2: Hardware architecture for edge detection algorithm

First a video frame A.1 is captured by the camera and the image data is stored in memory. The video frame A.1 is converted from RGB (Red, Green, Blue) format to grayscale, then edge detection is applied, the edges (in black color) are stored in memory and the video frame A.1 is displayed on the monitor. The system should wait a time, for example 2 seconds, to perform a new video frame capture. After waiting time a new video frame B.1 is captured by the camera and the image data is stored in memory. The video frame B.1 is converted from RGB format to grayscale, then edge detection is applied, the edges (in black color) are stored in memory. The image C.1 is generated from the matching pixels between the image A.1 and the image B.1, the image data C.1 are stored in memory.

After waiting time a new video frame A.2 is captured by the camera and the image data is stored in memory. The video frame A.2 is converted from RGB format to grayscale, then edge detection is applied, the edges (in black color) are stored in memory. The B.2 image is generated from the matching pixels between the A.2 and C.1 images. If there is a difference between the black pixels of A.2 and C.1, then there is a new object in the image and these pixels must be stored in the B.2 image, the pixel values must be incremented by a preconfigured value.

After waiting time a new video frame A.3 is captured by the camera and the image data is stored in memory. The video frame A.3 is converted from RGB format to grayscale, then edge detection is applied, the edges (in black color) are stored in memory. The B.2 image is updated from the matching pixels between the A.3 and C.1 images. If there is a difference between the black pixels of A.3 and C.1, then there is a new object in the image and these pixels must be stored in the B.2 image, the pixel values must be incremented by a preconfigured value. The part of the process described so far can be seen in Figure 3.

In the next step is verified if any pixel of the B.2 image has reached a pre-set threshold value, in the negative case the system must capture a new video frame by repeating the process until the threshold is reached. When the pixel value of the B.2 image reaches the preset threshold value is performed a merge between the last captured video frame A.n and the B.2 image, only for the pixels of the B.2 image that have been incremented. Finally the generated image is displayed and the system waits for a reset signal.

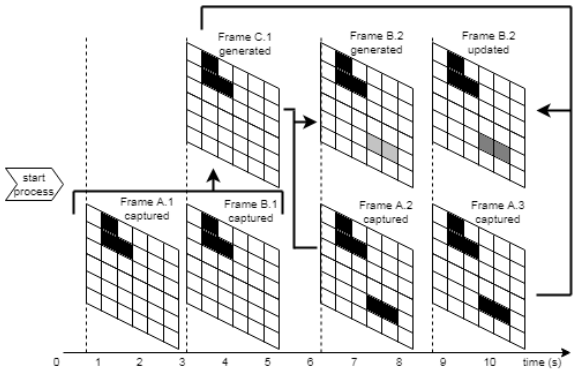


Figure 3: Object detection algorithm part 1

The entire process previously cited can be seen in Figure 4.

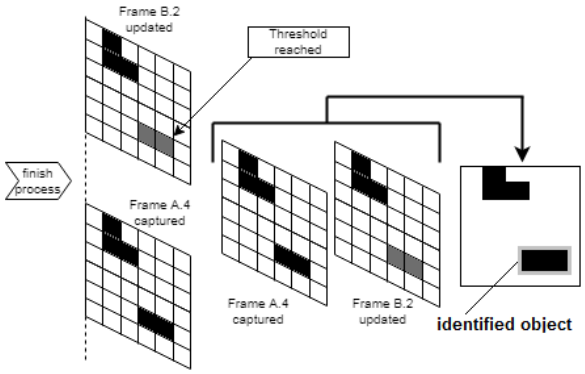


Figure 4: Object detection algorithm part 2

3.4 Image Acquisition Process

The video frame data is captured by the camera TRDB-D5M Terasic CMOS sensor and is forwarded to the Image Capture Module, where valid pixel information is extracted. The video frames captured by the camera are presented according to the Bayer pattern, for this reason a module is used to convert to RGB format.

The system can receive video frames with the resolution of 800x600 or 640x480 pixels, however a preconfiguration of the module that interfaces with the camera is required.

3.5 Hardware Implementation

In this work, the Verilog hardware description language was used to model the hardware. The proposed architecture (Figure 5) consists of 11 modules implemented and 5 IPs (intellectual property) modules that have been configured according to the project requirements.

- The FSM Control module is a finite state machine that coordinates operations between all system modules. This block receives and generates control signals that reinitialize, initialize, and execute operations on all processing modules.
- The Input Handler Ctrl module reads video frame data from RAW2RGB (IP) module, converts received pixels to the 8-bit size, and writes data to SDRAM.
- The RGB2GRAY Ctrl module reads video frame data from SDRAM, organizes and sends them to RGB2GRAY CONV module. The RGB2GRAY CONV module converts image pixels in RGB format to grayscale.
- The EDGE DETECT CTRL module coordinates operations between the EDGE DETECT and EDGE STORAGE modules. In addition, it adds a border to the video frame data, before edge detection processing. The EDGE DETECT module implements the edge detection algorithm, using the Sobel operator and preconfiguring a threshold as a parameter. The EDGE STORAGE module reads processed data by the EDGE DETECT module, organizes and writes data in the SDRAM.
- The Compare Pixel CTRL module implements the fixed object detection algorithm. This block reads/writes data in SDRAM.
- The CCD Capture module is a Soft IP Core which extracts valid pixels from the interface with the camera. The RAW2RGB module is a Soft IP Core which converts data from camera in Bayer Pattern format to the RGB format. The I2C CCD CONFIG module is a Soft IP Core which configures the camera parameters through configuration registers.
- The VGA CTRL module is a Soft IP Core which reads data from a video frame from SDRAM memory and synchronizes data with VGA interface.

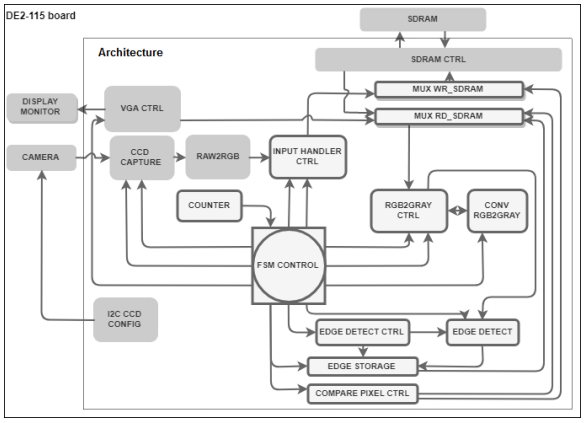


Figure 5: Hardware Architecture

3.6 Memory Organization

The SDRAM memory available on the Intel DE2-115 FPGA was used to store a captured video frame at 800x600 or 640x480 resolution. The SDRAM memory consists of 4 banks (only 2 banks are used in the project), each bank is a structured array of rows and columns (8192 rows x 512 columns x 16 bits). The access for data reading and writing from and to the SDRAM are made through bursts by the SDRAM Controller. Figure 6 presents a memory map with address space definitions.

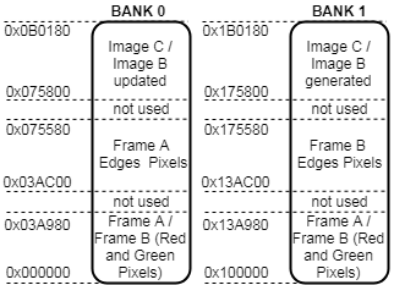


Figure 6: Memory Map

Table 1: Synthesis Result for FPGA

Elements	Utilization
Total logical elements	3,884 / 114,480 (3%)
Registers	2021
Total pins	428 / 529 (81%)
Total memory bits	98,360 / 3,981,312 (2%)
9-bit embedded multipliers	2 / 532 (<1%)
Total PLLs	1 / 4 (25%)

4 RESULTS

The module validation was performed through simulation using the Incisive Simulator verification tool from Cadence Design Systems. After the functional verification in simulation, the RTL (Register Transfer Level) code was synthesized for the Intel DE2-115 FPGA - Cyclone IV, device identifier EP4CE115F29C7. Table 1 shows the results obtained in the logic synthesis stage.

The results were obtained from FPGA prototyping. The tests includes TRDB-D5M Terasic CMOS sensor, the FPGA Intel DE2-115 device and the VGA (Video Graphics Array) monitor, to display the results.

The test scenario used two objects, when the system is started, only one object is static in the test scenario, as can be seen in the left side of Figure 7. This image can be considered as the first frame video captured. In the second video frame captured, two objects can be seen in the test scenario, in the right side of Figure 7. In this test scenario, all later captured video frames present a same configuration with two static objects, as obtained in the second captured video frame. Therefore, the system continues to capture new video frames until the threshold is reached, as can be seen on the left side of Figure 8.

Figure 7: 1st and 2nd video frames captured

After the control system detects that the threshold has been reached, the identified object is highlighted in the stored image, as can be seen on the right side of Figure 8. Finally the image is displayed on the VGA monitor.

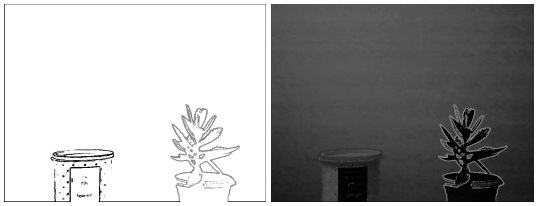


Figure 8: Edge detection and Object Detection

5 CONCLUSION

In this paper, we propose an FPGA design that provides a Real-Time Processing System for Object Detection in Digital Videos. The proposed architecture is designed and implemented on an Intel DE2-115 FPGA platform. Functional verification and FPGA prototyping verified the operation and integration of the entire proposed system. In future work, a digital filter implementation is planned to eliminate noise that has been identified during object detection process. We also intend to integrate the system with other communication protocols in order to increase the functionalities.

REFERENCES

[1] Gretchel Karen L Alcantara, Ivan Darren J Evangelista, Jerome Vincent B Malinao, Ofelia B Ong, Reginald Steven DM Rivera, and Engr Leonard U Ambata. 2018. Head Detection and Tracking Using OpenCV. In *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*. IEEE, 1–5.

[2] G Chandan, Ayush Jain, Harsh Jain, et al. 2018. Real Time object detection and Tracking using Deep Learning and openCV. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, 1305–1308.

[3] Rafael C Gonzalez and Richard E Woods. 2017. *Digital Image Processing*. Pearson.

[4] Kruti Goyal, Kartikey Agarwal, and Rishi Kumar. 2017. Face detection and tracking: Using OpenCV. In *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, Vol. 1. IEEE, 474–478.

[5] Souhail Guennouni, Ali Ahaitouf, and Anass Mansouri. 2014. Multiple object detection using OpenCV on an embedded platform. In *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*. IEEE, 374–377.

[6] Rahul R Palekar, Sushant U Parab, Dhrumil P Parikh, and Vijaya N Kamble. 2017. Real time license plate detection using openCV and tesseract. In *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2111–2115.

[7] S Yogitha and P Sakthivel. 2014. A distributed computer machine vision system for automated inspection and grading of fruits. In *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*. IEEE, 1–4.