Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Eletrônica

Implementação do Filtro Sobel em SoC FPGA utilizando algoritmo CORDIC

Autor: Victor Bastos Imbrosio Oliveira
Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF
2023

Victor Bastos Imbrosio Oliveira

Implementação do Filtro Sobel em SoC FPGA utilizando algoritmo CORDIC

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2023

Victor Bastos Imbrosio Oliveira
Implementação do Filtro Sobel em SoC FPGA utilizando algoritmo CORDIC/ Victor Bastos Imbrosio Oliveira. – Brasília, DF, 2023-
49 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

1. Arquitetura de Hardware. 2. Filtro Sobel. I. Prof. Dr. Daniel Mauricio Muñoz Arboleda. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação do Filtro Sobel em SoC FPGA utilizando algoritmo CORDIC

CDU 02:141:005.6

Victor Bastos Imbrosio Oliveira

Implementação do Filtro Sobel em SoC FPGA utilizando algoritmo CORDIC

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, DF, 10 de Abril de 2023:

Prof. Dr. Daniel Mauricio Muñoz Arboleda
Orientador

Prof. Dr. Diogo Caetano Garcia (FGA/UnB)
Convidado 1

Prof. Dr. Gilmar Silva Beserra (FGA/UnB)
Convidado 2

Brasília, DF
2023

Este trabalho é dedicado a todos aqueles que tiveram algum impacto na minha jornada acadêmica, desde amigos até professores, mas especialmente à minha amada mãe, que é meu pilar de apoio e um dos meus maiores exemplos.

Agradecimentos

Gostaria de agradecer primeiramente a minha mãe, Luiza Cristina Bastos, por estar comigo em todos os momentos, sejam eles bons ou ruins, sempre me oferecendo força, carinho e me inspirando a ter a coragem necessária para desbravar os desafios desse mundo. Agradeço também aos meus amigos, os quais me auxiliaram tanto emocionalmente quanto intelectualmente nessa aventura, fazendo da graduação uma das experiências mais enriquecedoras que eu já tive na vida. Agradeço ao meu orientador, Prof. Dr. Daniel Mauricio Muñoz Arboleda, por toda sua paciência e sabedoria ao lidar com minhas dúvidas, angústias e problemas durante essa jornada. Por fim, agradeço à Universidade de Brasília e aos seus servidores que, através de árduo trabalho, proporcionam minha admiração pelo acesso à educação pública gratuita e de qualidade.

"Nas minhas palavras, você não prestou atenção?
Passe adiante o que você aprendeu. Força. Maestria.
Mas fraqueza, tolice e fracasso também.
Sim, fracasso mais do que tudo.
O maior professor, o fracasso é."
(Mestre Yoda, Star Wars: Episódio VIII)

Resumo

A detecção de objetos é uma das tecnologias mais relevantes da área de visão computacional, podendo ser utilizada em diversas aplicações para diferentes áreas. Por vezes, essas aplicações necessitam de execução em tempo real e boa acurácia nos resultados gerados, o que torna pertinente o uso de hardware especializado. Este trabalho apresenta uma proposta de arquitetura de hardware do filtro Sobel que utiliza o algoritmo CORDIC para calcular a magnitude e direção do gradiente utilizando precisão de ponto fixo, com a finalidade de compor uma futura aplicação de detecção de objetos. A arquitetura foi desenvolvida levando em conta os benefícios da paralelização de operações, possível no desenvolvimento em hardware, a fim de acelerar ao máximo o algoritmo. Uma simulação comportamental e implementação em hardware demonstraram a capacidade da arquitetura de atender à exigência de execução em tempo real, mantendo uma boa exatidão através de um baixo erro quadrático médio.

Palavras-chave: Arquitetura de Hardware, FPGA, VHDL, Gradiente, Filtro Sobel, CORDIC.

Abstract

Object detection is one of the most relevant technologies in the area of computer vision, being used in several applications for different areas. Occasionally, these applications are submitted to real-time constraints and need good accuracy in the generated results, which makes the use of specialized hardware pertinent. This work presents a Sobel filter hardware architecture that uses CORDIC algorithm to calculate the magnitude and direction of the gradient using fixed-point precision, with the aim of composing a object detection application. The architecture was developed taking into account the benefits of parallel operations, possible in hardware development, in order to accelerate the algorithm as much as possible. A behavioral simulation and hardware implementation demonstrated the architecture's ability to meet real-time execution requirements while maintaining good accuracy through low mean squared error.

Keywords: Hardware Architecture, FPGA, VHDL, Gradient, Sobel Filter, CORDIC.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

CNN	*Convolutional Neural Network*
HOG	*Histogram of Oriented Gradients*
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit*
FPGA	*Field Programmable Gate Array*
CORDIC	*Coordinate Rotation Digital Computer*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very-High-Speed Integrated Circuit*
SoC	*System-On-Chip*
LUT	*Look-Up Table*
CLB	*Configurable Logic Block*
BRAM	*Block Random-Access Memory*
DSP	*Digital Signal Processor*
SRAM	*Static Random-Access Memory*
RAM	*Random-Access Memory*
AMBA	*Advanced Microcontroller Bus Architecture*
AXI	*Advanced eXtensible Interface*
ASIC	*Application Specific Integrated Circuit*
CMOS	*Complementary Metal–Oxide–Semiconductor*
FIFO	*First in First out*
FF	*Flip-Flops*
I/O	*Input/Output*
ROM	*Read-only Memory*
ILA	*Integrated Logic Analyzer*
IPCORE	*Intellectual Property Core*

Lista de símbolos

KB	Kilobyte
GB	Gigabyte
MHz	Mega-hertz
GHz	Giga-hertz
ms	Milissegundo
us	Microsegundo
ns	Nanosegundo
MP/s	Megapixeis por segundo
W	Watts
mW	Miliwatts

Sumário

1 Introdução

Visão computacional é o campo da ciência focado na criação de sistemas digitais capazes de processar, analisar e extrair informações importantes a partir de imagens, vídeos ou qualquer outro dado visual, tomando decisões baseadas nessas informações (IBM, s.d.). Com o objetivo de enxergar o mundo de forma análoga ao que os humanos são capazes de fazer, esses sistemas utilizam um dispositivo sensor capaz de captar dados visuais e um dispositivo interpretador capaz de dar sentido aos dados recebidos (BABICH, 2020). Veículos autônomos, reconhecimento facial, auxílio no setor de saúde, agricultura inteligente e automação industrial são apenas alguns exemplos de aplicações geradas a partir da área de visão computacional (MARR, 2019).

A detecção de objetos, oriunda da área de visão computacional, é uma técnica que permite a identificação e localização de instâncias de objetos contidos em imagens digitais e vídeos, de acordo com uma classificação específica, que abrange desde pessoas até veículos e alimentos (DASIOPOULOU et al., 2005). Implementações mais modernas dessa técnica costumam ser baseadas em dois princípios distintos, as que utilizam aprendizado de máquina e as que fazem uso de aprendizado profundo. Os métodos que utilizam aprendizado de máquina necessitam realizar a extração de recursos específicos da imagem, como histograma de cores ou bordas, para identificar grupos de pixels que podem pertencer a um objeto, inserindo essas informações em um modelo que faz a predição da localização do objeto, assim como sua classificação (FRITZ, 2021). Os métodos que empregam aprendizado profundo fazem uso de CNNs (*Convolutional Neural Networks*), sendo capazes de realizar a detecção não-supervisionada de objetos, do começo ao fim, sem definir recursos específicos (GIRSHICK et al., 2014).

Para detectores de objeto baseados em aprendizado de máquina, é muito comum a utilização de filtros para realizar a definição dos recursos específicos necessários, ao exemplo do algoritmo HOG (*Histogram of Oriented Gradients*). Este algoritmo é comumente utilizado para captar recursos humanos em imagens, retirando a aparência e forma de um objeto local, com a ideia de que esses recursos podem ser caracterizados de forma satisfatória através da análise da distribuição de gradientes de intensidade local e suas direções (DALAL; TRIGGS, 2005). Para realizar o cálculo da magnitude e direção do gradiente, necessário para o algoritmo HOG, pode-se utilizar o filtro Sobel para detectar bordas na imagem. Bordas são recursos fundamentais de uma imagem, sendo necessárias para determinar os formatos dos objetos contidos na mesma.

1.1 Descrição do Problema e Justificativa

Aplicações na área de visão computacional apresentam grande variedade de requisitos, com algumas necessitando alta velocidade de execução em seu processamento de dados, enquanto outras não. Como exemplo comparativo, pode-se pensar no uso de um algoritmo de detecção de objetos em dois cenários distintos: um em exames médicos para auxiliar radiologistas no diagnóstico de seus pacientes e outro em uma aplicação de veículos autônomos. O primeiro caso não necessita ser executado em tempo real, já que o diagnóstico resultante não se invalida caso seja entregue alguns minutos depois do começo do exame. O mesmo não pode ser dito para a segunda aplicação, que necessita de agilidade no acesso aos resultados, com tempo suficiente para realizar a detecção de veículos e pedestres em situações de tempo crítico, a fim de evitar colisões e tragédias.

Figura 1 – Visão geral das etapas do algoritmo HOG. Adaptado de: (DALAL; TRIGGS, 2005).

Na Figura 1 são apresentadas as etapas necessárias para a execução do algoritmo que implementa o HOG. A etapa de computação dos gradientes, o foco deste trabalho, consiste na aplicação de algum filtro que faça esta tarefa, como o filtro Sobel.

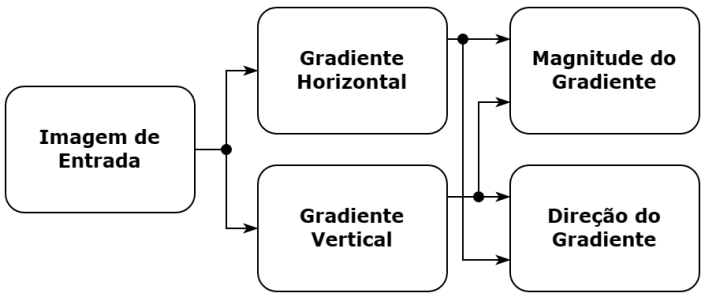


Figura 2 – Visão geral das etapas do filtro Sobel. Fonte: Elaboração Própria.

A Figura 2 apresenta a estrutura do filtro Sobel. Para realizar o cálculo dos gradientes nas direções horizontal e vertical, é necessário convoluir a imagem de entrada com o *kernel* específico do filtro. Essa convolução é dada através de diversas operações de soma, subtração e multiplicação para cada pixel analisado. Após isso, é necessário calcular a magnitude e direção do gradiente, o que introduz operações matemáticas como raiz quadrada e arco tangente. Devido a quantidade e complexidade das operações matemáticas envolvidas no processo, o filtro Sobel acaba se tornando um algoritmo de alto custo computacional, o que pode ser agravado caso a imagem de entrada seja de alta resolução.

Em aplicações de software, um programa que implementa este filtro roda através de uma CPU (*Central Processing Unit*) genérica, o que adiciona mais sobrecarga ao

processo, já que a arquitetura executa outras tarefas do sistema operacional em conjunto do algoritmo. Dessa forma, a implementação em software deste algoritmo se torna inadequada para aplicações onde a execução em tempo real e a portabilidade do sistema são requisitos necessários.

Uma solução para estes problemas reside na implementação desse filtro em arquitetura de hardware com processamento paralelo, como GPUs (*Graphics Processing Unit*) e dispositivos FPGA (*Field Programmable Gate Arrays*). Na literatura, as FPGAs costumam ser as plataformas mais utilizadas, devido à natureza das aplicações de detecção de objetos que geralmente englobam sistemas embarcados e se beneficiam da alta taxa de transferência, portabilidade, configurabilidade e baixo consumo de energia desses dispositivos (GHAFFARI et al., 2019). Essas implementações buscam estudar o algoritmo para conseguir identificar atividades que possam ser executadas simultaneamente, a fim de otimizar a velocidade do algoritmo levando em conta a capacidade de paralelizar processos intrínseca do desenvolvimento em hardware. Em contrapartida, muitos desses trabalhos não analisam a direção do gradiente, se preocupando apenas com a magnitude para a detecção de bordas. A precisão dos resultados no geral também não é analisada, sendo bastante comum utilizar aproximações matemáticas para trabalhar apenas com números inteiros. O presente trabalho pretende investigar essas lacunas, realizando uma análise de temporização, precisão de resultados e consumo de recursos em hardware.

1.2 Objetivos Gerais

O objetivo deste trabalho é implementar uma arquitetura de hardware do filtro Sobel que utiliza o algoritmo CORDIC (*Coordinate Rotation Digital Computer*) para calcular a magnitude e orientação do gradiente, realizando a caracterização dessa arquitetura, assim como sua validação através da comparação de seus resultados com um modelo de referência feito em software.

1.3 Objetivos Específicos

- Desenvolvimento de um modelo de referência em software que realize o cálculo da magnitude e direção do gradiente de uma imagem através do filtro Sobel, para realizar comparações com a arquitetura de hardware proposta.
- Desenvolvimento e implementação em VHDL (*VHSIC Hardware Description Language*) de uma arquitetura de hardware do filtro Sobel, com a finalidade de calcular a magnitude e direção do gradiente de uma imagem.
- Parametrizar a arquitetura em termos da resolução da imagem de entrada.

- Caracterização, simulação e validação da implementação física da arquitetura mapeada em um SoC (*System-On-Chip*) FPGA.

1.4 Organização do documento

O presente documento está organizado da seguinte maneira: o Capítulo 2 discorre sobre a fundamentação teórica que dá base para a implementação do filtro Sobel, destrinchando em detalhes as operações matemáticas aplicadas no filtro e suas finalidades, além de apresentar o estado da arte desse assunto, com trabalhos que já implementaram o filtro Sobel em hardware. O Capítulo 3 trata sobre a metodologia envolvida na implementação da arquitetura de hardware proposta para o filtro Sobel, entrando em detalhes sobre os sinais utilizados na arquitetura e suas funções. No Capítulo 4 são apresentados os resultados obtidos da implementação da arquitetura, assim como os resultados obtidos na simulação comportamental da mesma. Por fim, o Capítulo 5 apresenta a conclusão do documento, com discussões sobre os resultados obtidos.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos teóricos que fundamentam a implementação do filtro Sobel com o algoritmo CORDIC, destrinchando em detalhes as equações matemáticas aplicadas e suas finalidades. Nesta parte também é brevemente apresentado o SoC FPGA utilizado para implementar a arquitetura de hardware proposta, além de serem revisados alguns trabalhos que já implementaram o filtro Sobel em hardware.

2.1 Gradiente da Imagem

Em uma imagem, as bordas contidas na mesma são um dos recursos mais relevantes que podem ser extraídos. Este recurso é necessário para identificar estruturas e propriedades de objetos em uma cena, sendo frequentemente utilizado como primeiro passo para a recuperação de informações em uma imagem.

Para descobrir a intensidade e direção de uma borda em um local qualquer de uma imagem é apropriado realizar o uso da função gradiente. O gradiente de uma imagem é expresso como um vetor, no qual a magnitude mede a mudança de intensidade, enquanto a direção informa em qual sentido a imagem muda mais rapidamente (JACOBS, 2005). Partindo do princípio que uma borda ocorre quando há uma descontinuidade na função de intensidade ou um gradiente de intensidade muito acentuado na imagem, é possível realizar a detecção de bordas através da localização do ponto máximo da derivada do valor de intensidade na imagem (VINCENT; FOLORUNSO, 2009).

O vetor gradiente aponta na direção de máxima mudança de f em (x,y), sendo válido em um único ponto qualquer da imagem. Essa função, quando avaliada em todos os valores de x e y, tem cada elemento desse vetor dado pela Equação (2.1) (GONZALEZ; WOODS, 2018).

$$\nabla f(x,y) \equiv grad[f(x,y)] \equiv \begin{bmatrix} g_x(x,y) \\ g_y(x,y) \end{bmatrix} \equiv \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} \tag{2.1}$$

A magnitude do vetor gradiente em um ponto da imagem é dada pela sua norma Euclidiana, como descrito na Equação (2.2). O resultado desse cálculo atribui um valor para a taxa de mudança na direção do vetor gradiente em um ponto (x,y). Para descrever a direção do vetor gradiente nesse mesmo ponto, é calculado o ângulo do vetor gradiente, utilizando a Equação (2.3) (GONZALEZ; WOODS, 2018).

$$M(x,y) = ||\nabla f(x,y)|| = \sqrt{g_x^2(x,y) + g_y^2(x,y)} \tag{2.2}$$

$$\alpha(x, y) = \arctan\left(\frac{g_y(x, y)}{g_x(x, y)}\right) \tag{2.3}$$

Com a finalidade de obter o gradiente da imagem, é necessário computar as derivadas parciais em x e y para cada pixel da imagem. Dessa forma, é comum que a técnica de diferenças finitas, mais especificamente diferenças finitas progressivas, seja utilizada para esse cálculo (GONZALEZ; WOODS, 2018). As Equações (2.4) e (2.5) descrevem essa técnica, sendo implementada pelos *kernels* unidimensionais da Figura 3.

$$g_x(x, y) = \frac{\partial f(x, y)}{\partial x} = f(x + 1, y) - f(x, y) \tag{2.4}$$

$$g_y(x, y) = \frac{\partial f(x, y)}{\partial y} = f(x, y + 1) - f(x, y) \tag{2.5}$$

(a) *Kernel* gerado da Equação (2.4). (b) *Kernel* gerado da Equação (2.5).

Figura 3 – *Kernels* unidimensionais. Fonte: Elaboração Própria.

2.1.1 Operador Sobel-Feldman

Apresentado em 1968 por Irwin Sobel e Gary Feldman, no Projeto de Inteligência Artificial de Stanford, o operador Sobel-Feldman surgiu a partir da motivação de desenvolver uma estimativa de gradiente com computação eficiente e mais isotrópica que outros operadores para detecção de bordas utilizados na época (SOBEL, 2014). Dessa forma, foi idealizado um *kernel* bidimensional com centro simétrico, o qual seria capaz de extrair mais informações sobre bordas em imagens, já que o cálculo final do gradiente seria influenciado por todos os pixels na imediata vizinhança do pixel central. Para isso foi utilizada a técnica de diferenças finitas, descrita nas Equações (2.4) e (2.5), aplicada a uma janela qualquer de uma imagem, exemplificada na Figura 4, resultando nas Equações (2.6) e (2.7).

$$g_x = \frac{\partial f}{\partial x} = (p_7 + p_8 + p_9) - (p_1 + p_2 + p_3) \tag{2.6}$$

$$g_y = \frac{\partial f}{\partial y} = (p_3 + p_6 + p_9) - (p_1 + p_4 + p_7) \tag{2.7}$$

p_1	p_2	p_3
p_4	p_5	p_6
p_7	p_8	p_9

Figura 4 – Janela 3x3 arbitrária de uma imagem. Fonte: Elaboração Própria.

O operador Sobel-Feldman sugere uma variação das Equações (2.6) e (2.7), utilizando o dobro de peso nos coeficientes centrais, a fim de enfatizar os pixels relacionados diretamente ao pixel central em uma determinada direção. Essa modificação ajuda na suavização da imagem, tornando a filtragem mais resistente a ruídos (GONZALEZ; WOODS, 2018). As Equações (2.8) e (2.9) demonstram essa modificação, e a Figura 5 apresenta os *kernels* resultantes utilizados no filtro Sobel.

$$g_x = \frac{\partial f}{\partial x} = (p_7 + 2p_8 + p_9) - (p_1 + 2p_2 + p_3) \tag{2.8}$$

$$g_y = \frac{\partial f}{\partial y} = (p_3 + 2p_6 + p_9) - (p_1 + 2p_4 + p_7) \tag{2.9}$$

-1	-2	-1
0	0	0
1	2	1

(a) *Kernel* gerado da Equação (2.8).

-1	0	1
-2	0	2
-1	0	1

(b) *Kernel* gerado da Equação (2.9).

Figura 5 – *Kernels* do filtro Sobel. Fonte: Elaboração Própria.

Considerando a janela descrita na Figura 4, é possível aproximar o cálculo do gradiente vertical pela diferença entre a terceira e a primeira linha, como visto na Figura 5a, assim como do gradiente horizontal pela diferença entre a terceira e primeira coluna da janela estabelecida, descrito na Figura 5b. Nota-se que a soma dos coeficientes dos *kernels* descritos resulta em zero, o que se traduz para uma resposta nula em áreas de intensidade constante na imagem (GONZALEZ; WOODS, 2018).

2.1.2 Computação do Gradiente

Para computar o gradiente da imagem utilizando o operador Sobel-Feldman, é necessário realizar uma convolução dos *kernels* da Figura 5 com a imagem à ser tratada. Essa operação é feita deslizando os *kernels* pela imagem, colocando-os centralizados em relação ao pixel que será analisado, formando uma janela de pixels que interage com os *kernels* para o cálculo. A varredura da imagem pode ser feita em qualquer ordem, contanto que o *kernel* seja convoluído uma vez com cada pixel. Uma forma comum de realizar a varredura consiste em começar da esquerda para a direita, sempre abaixando uma linha após varrer todas as colunas da matriz que representa a imagem original. A Figura 6 exemplifica as três primeiras janelas de uma imagem qualquer, utilizando a varredura mencionada.

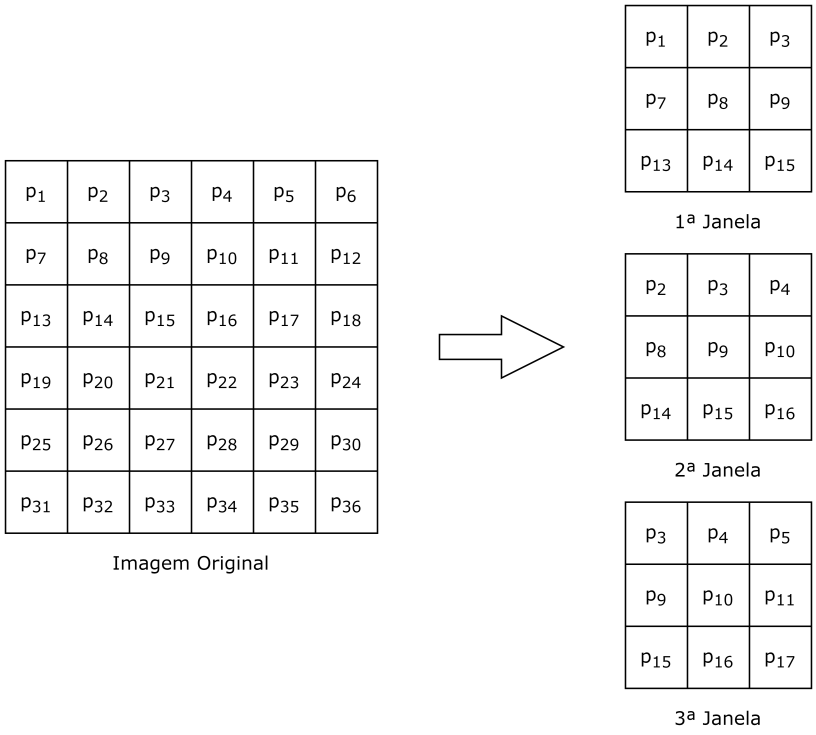
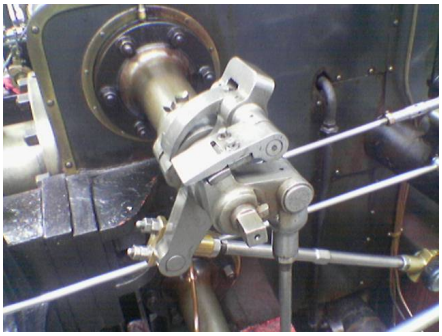
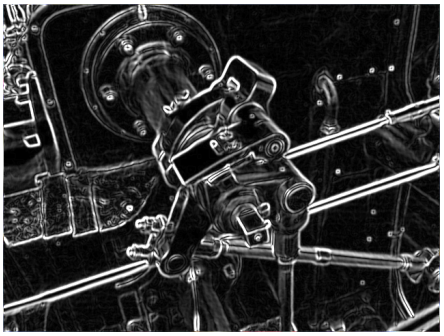


Figura 6 – Exemplo de três primeiras iterações da varredura da imagem para formação da janela. Fonte: Elaboração Própria.

Na convolução, multiplica-se cada um dos coeficientes da janela com a posição correspondente no *kernel* e em seguida soma-se todos os valores encontrados, resultando em apenas um valor de gradiente que é guardado em outra matriz. Com dois *kernels* de convolução, são geradas duas matrizes de gradiente, uma com gradientes horizontais e outra com gradientes verticais. Tendo ambas as matrizes de gradiente preenchidas, é possível formar outras duas matrizes com a magnitude e direção do gradiente em cada ponto da imagem original, utilizando as Equações (2.2) e (2.3). Dessa forma, seguindo todos os passos da filtragem e dispondo a matriz de magnitude do gradiente, é gerada uma imagem de saída que contém as bordas realçadas, exemplificada na Figura 7.



(a) Imagem original.



(b) Imagem filtrada.

Figura 7 – Comparação entre entrada e saída do filtro Sobel. Fonte: Wikipedia.

2.2 Algoritmo CORDIC

Criado em 1956 por Jack Volder no departamento de eletrônica aeronáutica da Convair, o CORDIC é um algoritmo desenvolvido para execução em tempo real, maximizando sua eficiência em hardware através do uso de adições, subtrações, deslocamento de bits e LUTs (*Look-Up Tables*). A finalidade do algoritmo é realizar operações matemáticas computacionalmente complexas como raiz quadrada, multiplicação, divisão, funções trigonométricas, funções hiperbólicas, assim como exponenciais e logaritmos de base arbitrária (VOLDER, 1959). O algoritmo possui dois modos de operação: rotacional e vetorização. No modo rotacional, são dadas as componentes de coordenadas de um vetor, assim como um ângulo de rotação, que são utilizados para calcular as componentes de coordenadas resultantes após a rotação através do ângulo dado. Já no modo vetorização, os componentes de coordenadas de um vetor são fornecidos para calcular a magnitude e a direção desse vetor (VOLDER, 1959).

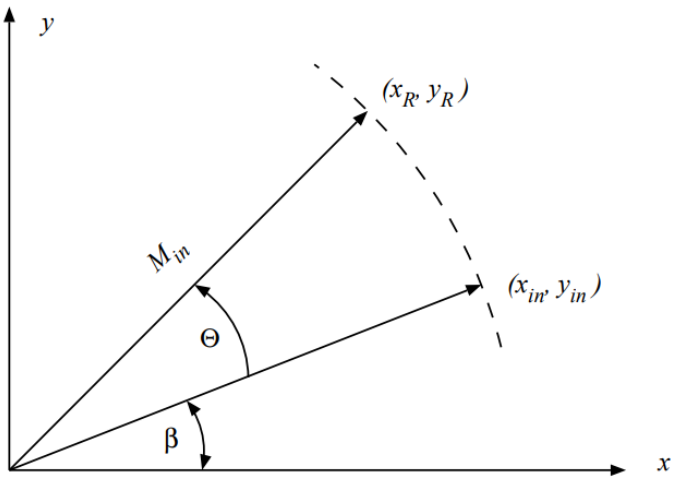


Figura 8 – Rotação de um vetor qualquer. Fonte: (ERCEGOVAC; LANG, 2003).

A Figura 8 apresenta a rotação de um vetor com coordenadas iniciais x_{in} e y_{in} para um vetor de coordenadas x_R e y_R, operação que pode ser descrita pelas Equações

(2.10) e (2.11). A variável M_{in} é o módulo do vetor, enquanto β é o valor do ângulo inicial. Essas equações podem ser expressas no formato matricial, resultando na Equação (2.12), que evidencia a matriz de rotação $ROT(\theta)$.

$$x_R = M_{in}\cos(\beta+\theta) = x_{in}\cos(\theta) - y_{in}\sin(\theta) \tag{2.10}$$

$$y_R = M_{in}\sin(\beta+\theta) = x_{in}\sin(\theta) + y_{in}\cos(\theta) \tag{2.11}$$

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} = ROT(\theta) \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{2.12}$$

O ângulo θ pode ser decomposto por ângulos de rotação elementares α_n, seguindo a Equação (2.13). Essa operação resulta em uma nova expressão para a matriz de rotação $ROT(\theta)$, que é descrita na Equação (2.14) (ERCEGOVAC; LANG, 2003).

$$\theta = \sum_{n=0}^{\infty} \alpha_n \tag{2.13}$$

$$ROT(\theta) = \prod_{n=0}^{\infty} ROT(\alpha_n) \tag{2.14}$$

Considerando a decomposição do ângulo θ como o somatório de infinitos ângulos de rotação elementares, sua substituição é feita nas Equações (2.10) e (2.11). Ao evidenciar o termo $\cos(\alpha_n)$ das fórmulas, são obtidas as Equações (2.15) e (2.16).

$$x_R[n+1] = x_R[n]\cos(\alpha_n) - y_R[n]\sin(\alpha_n) = \cos(\alpha_n)(x_R[n] - y_R[n]\tan(\alpha_n)) \tag{2.15}$$

$$y_R[n+1] = x_R[n]\sin(\alpha_n) + y_R[n]\cos(\alpha_n) = \cos(\alpha_n)(x_R[n] + y_R[n]\tan(\alpha_n)) \tag{2.16}$$

Com o intuito de evitar multiplicações no algoritmo e aplicar apenas deslocamentos de bits, escolhe-se um ângulo elementar baseado em potências de dois, como descrito na Equação (2.17), no qual $\sigma_n \in \{-1, 1\}$ (ERCEGOVAC; LANG, 2003).

$$\alpha_n = \arctan(\sigma(2^{-n})) = \sigma_n \arctan(2^{-n}) \tag{2.17}$$

Nota-se nas Equações (2.15) e (2.16) que o termo $\cos(\alpha_n)$ funciona como um ganho para o sistema. A Equação (2.18) demonstra este ganho após a substituição pela

Equação (2.17), no qual o termo σ_n desaparece devido ao seu valor elevado à dois ser sempre um unitário positivo. Sendo assim, é possível expressar este termo no final da somatória como um fator de escala, realizando a aproximação apresentada na Equação (2.19) (ERCEGOVAC; LANG, 2003).

$$M[n+1] = K[n]M[n] = \frac{1}{\cos(\alpha_n)}M[n] = \sqrt{1+\sigma_n^2 2^{-2n}}M[n] = \sqrt{1+2^{-2n}}M[n] \quad (2.18)$$

$$K = \prod_{n=0}^{\infty}\sqrt{1+2^{-2n}} \approx 1.6468 \quad (2.19)$$

Considerando a omissão do fator de escala, assim como a substituição da Equação (2.17) nas Equações (2.15) e (2.16), são obtidas as Equações (2.20) e (2.21). Para determinar o valor de σ_n é necessário guardar os valores de rotações anteriores a fim de comparar o valor total da rotação acumulada com o ângulo desejado, funcionando como uma realimentação negativa (ERCEGOVAC; LANG, 2003). Dessa forma, é possível incorporar esta mecânica no algoritmo, através da Equação (2.22).

$$x[n+1] = x[n] - \sigma_n 2^{-n} y[n] \quad (2.20)$$

$$y[n+1] = y[n] + \sigma_n 2^{-n} x[n] \quad (2.21)$$

$$z[n+1] = z[n] - \sigma_n \arctan(2^{-n}) \quad (2.22)$$

As Equações 2.20, 2.21 e 2.22 em conjunto implementam o algoritmo CORDIC. Para a aplicação com o filtro Sobel, é necessário utilizar o algoritmo em modo vetorização, a fim de obter o valor equivalente a magnitude na saída x e o valor da direção na saída z. Nesse modo, é necessário rotacionar o vetor inicial (x_{in}, y_{in}) até que o valor de y se aproxime de zero, enquanto se acumula o ângulo de rotação em z. Para um vetor inicial no primeiro quadrante, o termo σ_n segue a relação dada na Equação (2.23) (ERCEGOVAC; LANG, 2003).

$$\sigma_n = \begin{cases} 1, \text{ se } y[n] < 0 \\ -1, \text{ se } y[n] \geq 0 \end{cases} \quad (2.23)$$

Com os valores iniciais $x[0] = x_{in}$, $y[0] = y_{in}$ e $z[0] = z_{in}$, os valores finais do CORDIC são dados pelas Equações (2.24), (2.25) e (2.26) (ERCEGOVAC; LANG, 2003).

$$x_f = K\sqrt{x_{in}^2 + y_{in}^2} \quad (2.24)$$

$$y_f = 0 \tag{2.25}$$

$$z_f = z_{in} + \arctan\left(\frac{y_{in}}{x_{in}}\right) \tag{2.26}$$

2.3 Hardware Reconfigurável

FPGAs são dispositivos semicondutores que contém matrizes de blocos lógicos programáveis, conectados através de interconexões programáveis (XILINX, 2022a). A Figura 9 apresenta um exemplo de estrutura interna de uma FPGA, sendo o modelo apresentado pertercente a família Artix-7 da Xilinx.

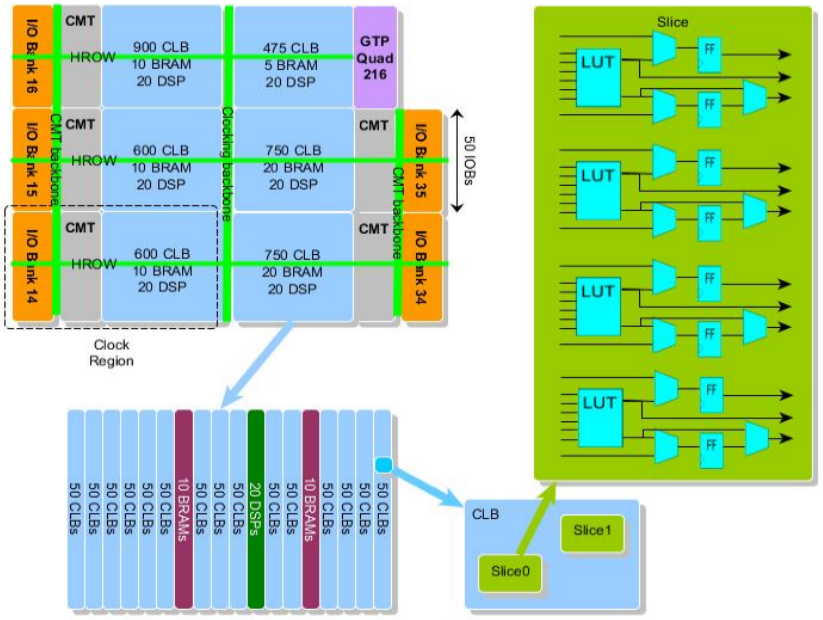


Figura 9 – Exemplo de estrutura interna de uma FPGA. Fonte: (EB, 2015).

A arquitetura da Figura 9 mostra a presença de vários CLBs (*Configurable Logic Block*), BRAMs (*Block Random-Access Memory*) e DSPs (*Digital Signal Processor*). A programação dos blocos é armazenada em *latches* internos do tipo SRAM (*Static Random-Access Memory*), com programação volátil ou não, dependendo do tipo de configuração escolhida. Como visto na Figura 9, cada CLB contém dois *slices*, que são agrupamentos de LUTs, multiplexadores e *flip-flops*. Os *slices* se dividem nos tipos SLICEL, que só pode ser configurado para operações combinacionais, e SLICEM, que além das funcionalidades do SLICEL, também pode ser configurado como um bloco de memória ou registrador de deslocamento. As BRAMs são vetores de memória que podem ser acessados com várias configurações de tamanho, além de poderem ser modificados para modos de leitura, escrita ou ambos. Por fim, os DSPs contém acumuladores e multiplicadores especializados para operações matemáticas.

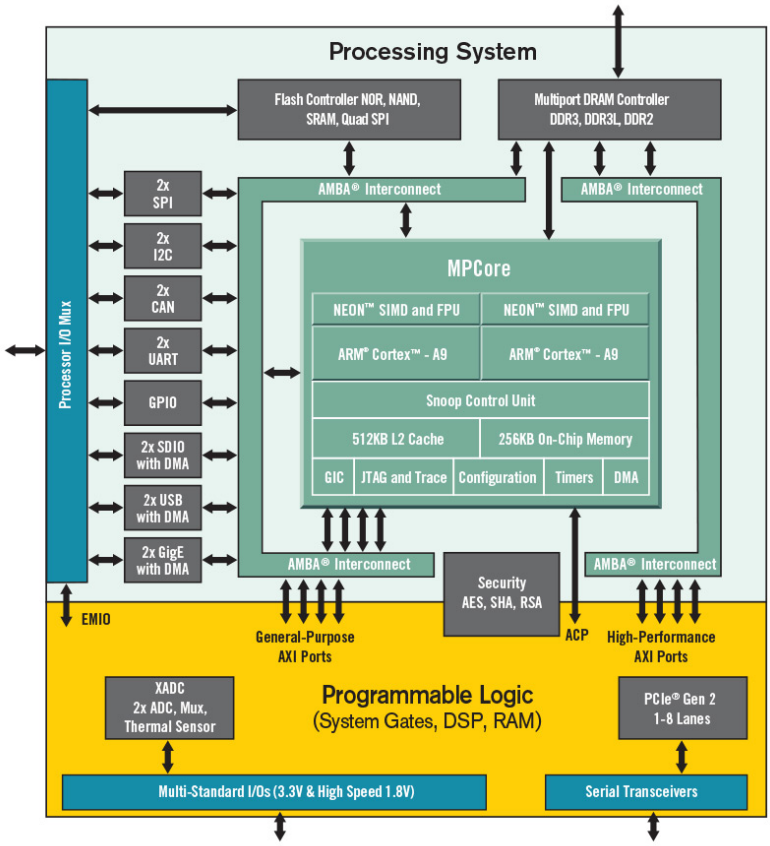


Figura 10 – Arquitetura do Zynq-7000 da Xilinx. Fonte: (XILINX, 2022b).

A Figura 10 apresenta a arquitetura geral do SoC FPGA Zynq-7000, o qual foi utilizado neste trabalho para a implementação da arquitetura de hardware proposta. O bloco superior da imagem mostra o sistema de processamento, que abriga dois processadores *single-core* ARM Cortex™-A9, uma memória cache L2 de 512 KBs e uma memória *on-chip* de 256 KBs. O bloco inferior da imagem mostra a parte da lógica programável, baseada em tecnologia Artix-7, contendo os DSPs, RAMs (*Random-Access Memory*) e portas lógicas necessárias para a implementação dos algoritmos. Esses blocos se conectam por meio do barramento AMBA (*Advanced Microcontroller Bus Architecture*), que abriga interfaces AXI (*Advanced eXtensible Interface*) de propósito geral e outras interfaces de alta performance.

A principal vantagem de fazer o uso de um SoC FPGA ao invés de ASICs (*Application Specific Integrated Circuit*) se dá no fato de que esse possue um ambiente mais adequado ao desenvolvimento de projetos que utilizem tanto hardware quanto software, cada vez mais comum em aplicações modernas, devido a sua arquitetura que integra processador e FPGA em apenas um dispositivo (INTEL, 2014). Dessa forma, é possível que um projeto se beneficie da alta velocidade e poder computacional oferecidos pelo desenvolvimento em hardware, com a flexibilidade e dinamismo do desenvolvimento em software.

2.4 Estado da Arte

Alguns trabalhos anteriores já exploraram a implementação do filtro Sobel em FPGA, com diversas finalidades. Uma dessas implementações foi feita em (NAUSHEEN et al., 2018), utilizando a placa Xilinx Spartan 6 XC6SLX43TQG144. Neste trabalho foi proposta uma arquitetura de oito bits com a finalidade de recuperar os endereços de pixels envolvidos no processo de convolução, reduzindo a complexidade do espaço neste processo.

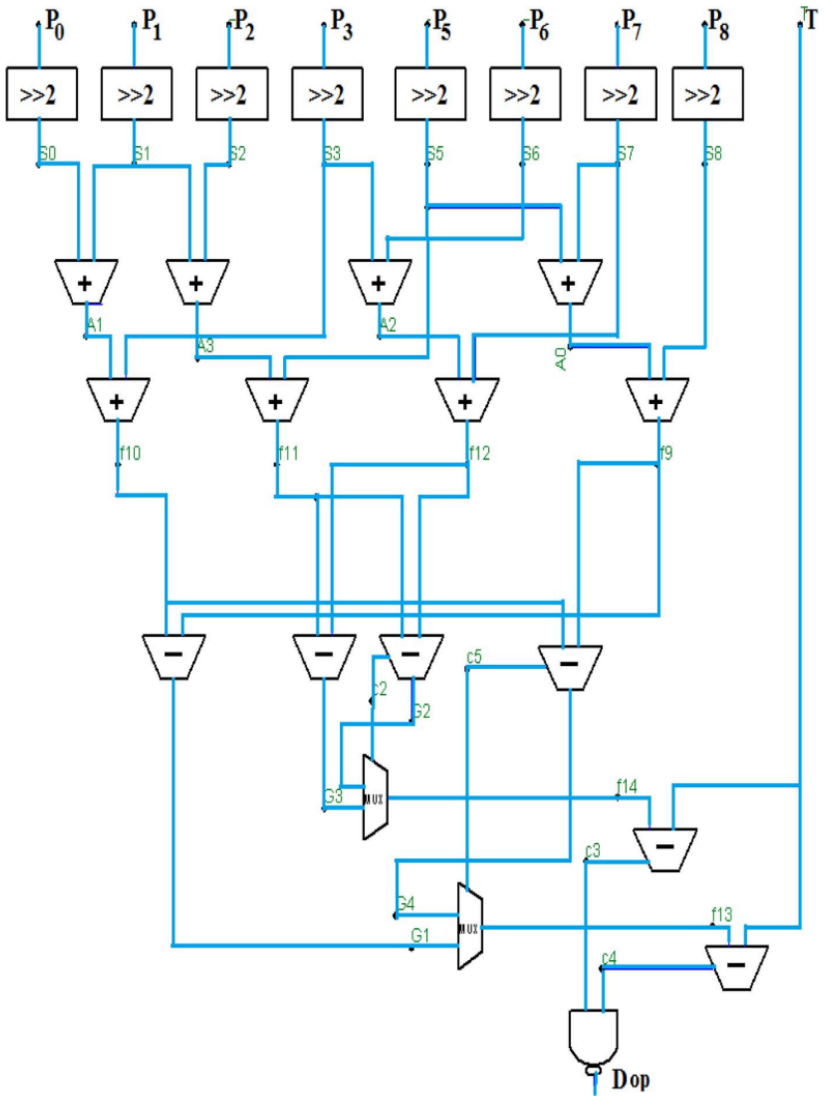


Figura 11 – Circuito para o filtro Sobel em FPGA. Fonte: (NAUSHEEN et al., 2018).

Na Figura 11, o circuito inicia deslocando os pixels da janela de convolução em dois bits à direita, efetivamente realizando uma divisão por quatro em todos os valores que chegam na estrutura, a fim de limitar a largura dos resultados da filtragem em oito bits. Os autores utilizam uma equação para extrair o índice de cada pixel da janela, fazendo com que não haja armazenamento de pixels redundantes, já que a extração de índices a partir do vetor de memória garante que cada novo pixel de entrada forme uma nova

janela de convolução, gerando um novo resultado. Após isso, são somados e subtraídos os valores de uma forma que seja feita a convolução dos pixels da janela com o *kernel* do filtro Sobel. Nos multiplexadores, são retirados os valores absolutos dos gradientes horizontal e vertical, para poder realizar o cálculo da magnitude do gradiente através de uma aproximação com números inteiros. Por fim, é feito um processo de limiarização com um valor arbitrário, através da porta lógica NAND.

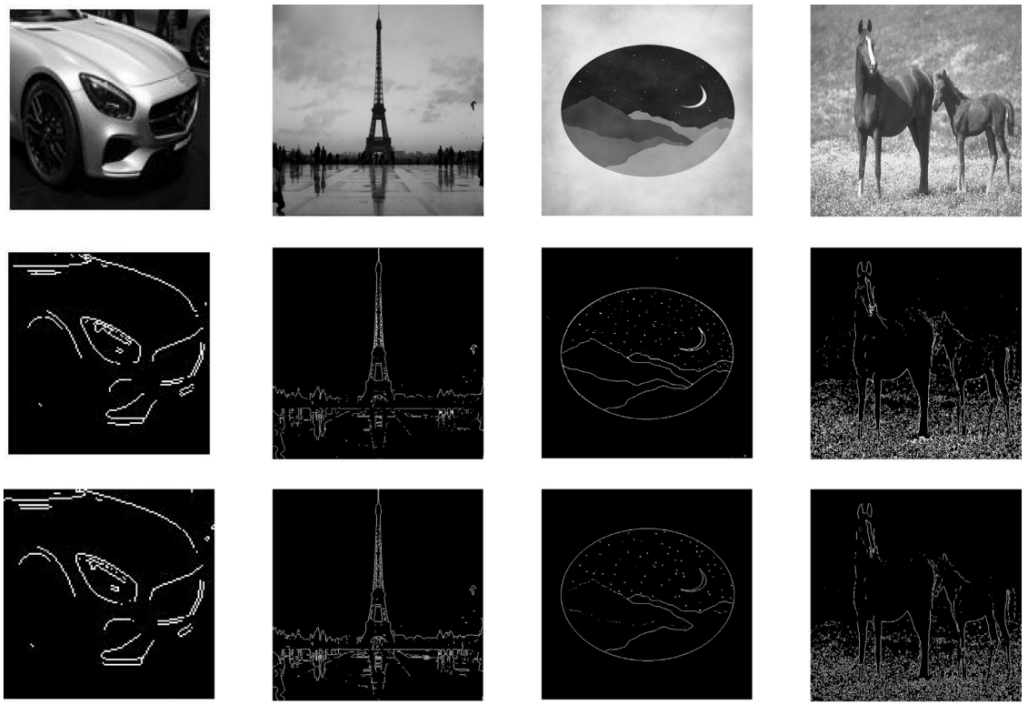


Figura 12 – Comparação visual entre implementações do filtro Sobel em hardware (segunda linha) e software (terceira linha). Fonte: (NAUSHEEN et al., 2018).

A Figura 12 apresenta a comparação dos resultados obtidos no trabalho, feito com imagens em escala de cinza com oito bits de largura e resoluções variadas. Os autores reportaram uma frequência máxima de operação de 504 MHz, com um tempo total de execução de 0,032 ms para imagens com resolução 128x128 e 0,52 ms para imagens de resolução 512x512. Não foram mostrados dados comparativos de precisão entre os resultados da arquitetura de hardware e os resultados em software.

No trabalho apresentado em (CHAPLE; DARUWALA, 2014), foram comparados os recursos consumidos em três placas diferentes utilizando a mesma arquitetura. Foram usadas as placas Xilinx Spartan-3 XC3S400 (1), Xilinx Spartan-6 XC6SLX25 (2) e Xilinx Vertex-5 XC5VLX50 (3). O objetivo deste trabalho foi implementar uma arquitetura de hardware do filtro Sobel capaz de realizar a filtragem em tempo real, considerando um quadro de um típico sensor câmera digital CMOS (*Complementary Metal–Oxide–Semiconductor*), com taxa de atualização de trinta quadros por segundo e frequência de saída para pixels válidos de 48 MHz.

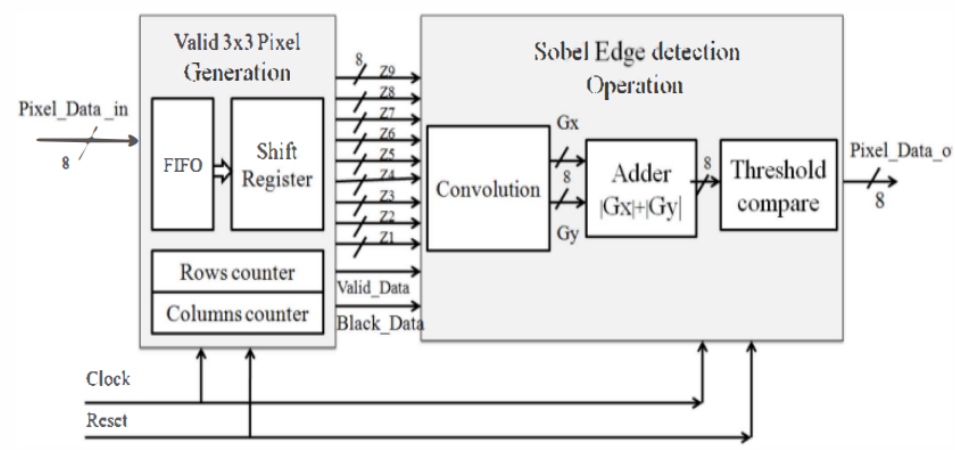


Figura 13 – Arquitetura para detecção de bordas utilizando filtro Sobel em FPGA. Fonte: (CHAPLE; DARUWALA, 2014).

Na Figura 13 é possível notar a presença de dois blocos principais. O bloco que gera a matriz 3x3 de pixels válidos é composto por dois vetores de memória arranjados no esquema FIFO (*First in, First out*) e três registradores de deslocamento, a fim de acumular os pixels corretos para a convolução. Nesse bloco também estão presentes um contador de linhas e um contador de colunas para realizar a identificação dos limites da matriz e preenchê-las com zero, fazendo com que a imagem de entrada tenha o mesmo tamanho da imagem de saída. No bloco que realiza a detecção de bordas com filtro Sobel, são realizadas as convoluções com as máscaras horizontal e vertical, adição do valor absoluto dos gradientes para a aproximação da magnitude e, por fim, um processo de segmentação binária resultando no pixel de saída da arquitetura.

Figura 14 – Comparação entre imagem original e imagem filtrada. Fonte: (CHAPLE; DARUWALA, 2014).

A Figura 14 mostra o resultado do teste feito com uma entrada em escala de cinza com oito bits e resolução de 640x480. Segundos os autores, a filtragem foi realizada dentro dos limites de tempo real pré-estabelecidos, com tempo total de execução de 7,696 ms rodando em uma frequência de 40 MHz e 6,412 ms em uma frequência de 48 MHz. A comparação de consumo de recurso entre as placas utilizadas pelos autores apontou a

placa da família *Vertex-5* como mais apropriada para este tipo de aplicação, devido a sua arquitetura mais moderna capaz de realizar operações matemáticas com mais eficiência.

Em (GUO; XU; CHAI, 2010), foi implementada uma arquitetura com *kernels* adicionais, baseados nos *kernels* originais do filtro Sobel. Ao total foram utilizados quatro *kernels* para retirada do gradiente nos ângulos 0º, 45º, 90º e 135º, a fim de aumentar a precisão do resultado final. A placa utilizada foi a Xilinx Spartan-3 XC3S200.

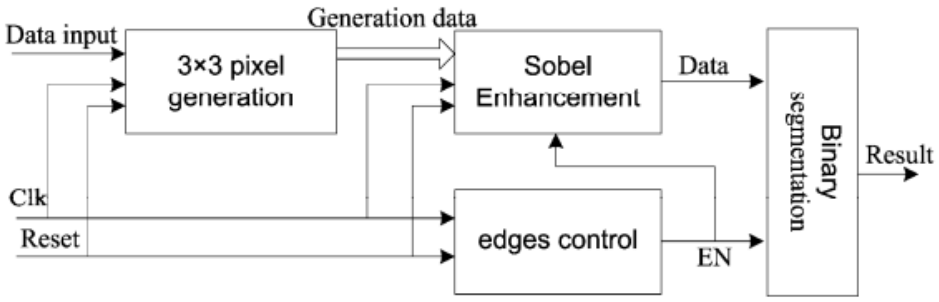


Figura 15 – Arquitetura para o filtro Sobel em FPGA. Fonte: (GUO; XU; CHAI, 2010).

A Figura 15 apresenta a arquitetura geral proposta, que foi dividida em quatro blocos menores. O bloco gerador da janela 3x3 de pixels é composto por três registradores de deslocamento e dois vetores no esquema FIFO, implementados a partir de uma memória RAM *dual-port*. No bloco de realce com filtro Sobel, os gradientes são calculados utilizando os *kernels* previamente citados e logo em seguida servem como entrada para um comparador, com a finalidade de levar para a saída apenas o gradiente de maior valor dentre os calculados. O bloco de controle das bordas serve para identificar quando um pixel da borda da imagem está em operação, a fim de zerar a saída devido as limitações do filtro Sobel. Por fim, o bloco de segmentação binária realiza a limiarização dos gradientes encontrados com um valor arbitrário, gerando o resultado.

Figura 16 – Comparação entre imagem original e imagem filtrada. Fonte: (GUO; XU; CHAI, 2010).

A Figura 16 apresenta o resultado obtido no trabalho, com um limiar zero. Os autores reportaram que para uma imagem de resolução 1024x1024 em escala de cinza de oito bits, o tempo total de execução foi de 21 ms, com uma frequência de 50 MHz para o sistema.

No trabalho (MUNOZ et al., 2017), o filtro Sobel foi implementado juntamente com um circuito localizador de picos, na tarefa de realizar a detecção de vibrações em fibra ópticas baseadas em reflectometria óptica no domínio do tempo sensível à fase. Para a implementação em hardware, foi utilizada uma FPGA Xilinx Zynq-7000.

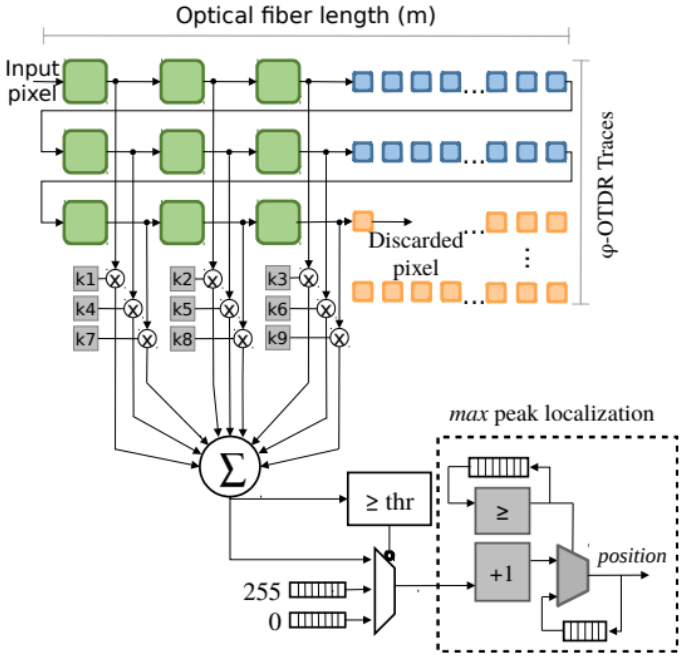


Figura 17 – Arquitetura para detecção de vibrações utilizando o filtro Sobel. Fonte: (MUNOZ et al., 2017).

A arquitetura mostrada na Figura 17, abriga uma estrutura de janelamento que utiliza um vetor no esquema FIFO para armazenar duas linhas e três pixels vindos da fibra óptica. Essa quantidade de pixels é suficiente para formar uma janela de convolução, pegando os índices dessa janela diretamente do vetor FIFO. Com isso, é feita a convolução através do filtro Sobel, realizando a multiplicação de cada índice com seu respectivo *kernel* e somando todos os resultados posteriormente. Esse resultado de gradiente passa por um processo de segmentação binária, com um limiar arbitrário. Por fim, este valor passa pelo circuito de detecção de picos, a fim de acusar o acontecimento de uma vibração na fibra. Vale ressaltar que a arquitetura de hardware proposta neste trabalho não soluciona o problema da convolução com uma janela 3x3 nos limites da imagem, tendo que utilizar um tratamento em software para corrigir esta etapa.

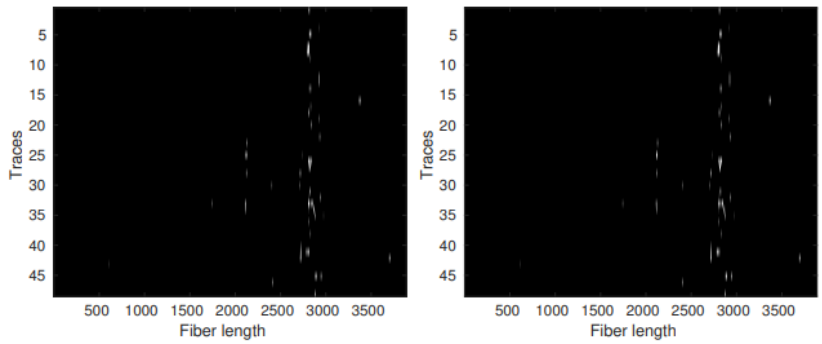


Figura 18 – Resultados obtidos do filtro Sobel em um ponto de perturbação na fibra óptica. Fonte: (MUNOZ et al., 2017).

A Figura 18 mostra o resultado obtido no trabalho para uma perturbação de um ponto na fibra óptica, com a imagem da esquerda sendo o resultado do modelo de referência em software, enquanto a imagem da direita sendo o resultado da arquitetura de hardware. Para uma amostra com resolução de 50x3890, os autores reportaram uma latência de 0,304 ms e uma taxa de transferência de 16 ns, o que resulta em um tempo total de execução de 3,416 ms.

Tabela 1 – Resumo dos recursos de hardware consumidos nos trabalhos revisados.

Trabalho	**Recursos de Hardware**				
	LUT	***FF***	***BRAM***	***DSP***	***I/O Pin***
(NAUSHEEN et al., 2018)	114	0	N/A	N/A	57
(CHAPLE; DARUWALA, 2014) (1)	3166	339	N/A	N/A	97
(CHAPLE; DARUWALA, 2014) (2)	1680	451	N/A	N/A	N/A
(CHAPLE; DARUWALA, 2014) (3)	1927	339	N/A	N/A	N/A
(GUO; XU; CHAI, 2010)	346	289	2	N/A	18
(MUNOZ et al., 2017)	344	457	11	0	N/A

Na Tabela 1 são comparados o consumo de recursos em termos de LUTs, FFs (*Flip-Flops*), BRAMs, DSPs e pinos de I/O (*Input/Output*). Nota-se que as arquiteturas que tiveram um maior uso de BRAM, foram também as arquiteturas que conseguiram minimizar o uso de LUTs, provavelmente devido à forma que o janelamento é organizado nestes trabalhos. Percebe-se também que todos os autores evitaram o uso de DSPs, já que esse recurso é complexo e não é necessário nas multiplicações do *kernel* do filtro Sobel, que podem ser aplicadas apenas com deslocamentos.

3 Metodologia

Neste capítulo é apresentada a arquitetura de hardware para o filtro Sobel proposta para este trabalho. A arquitetura foi organizada no esquema de *pipeline*, no qual uma próxima etapa só inicia após a confirmação da etapa anterior ter finalizado. Foi feita a combinação de uma ROM (*Read-only Memory*) com um módulo controlador, a fim de manejar a imagem de entrada utilizada pelo filtro Sobel. A resolução dessa imagem foi parametrizada na arquitetura, fazendo com que diferentes tamanhos de imagem pudessem ser utilizados. O ILA (*Integrated Logic Analyzer*), foi utilizado para analisar a saída do filtro pós-implementação no kit de desenvolvimento. A Figura 19 apresenta as conexões entre o controle de memória, a ROM, o filtro Sobel e o ILA.

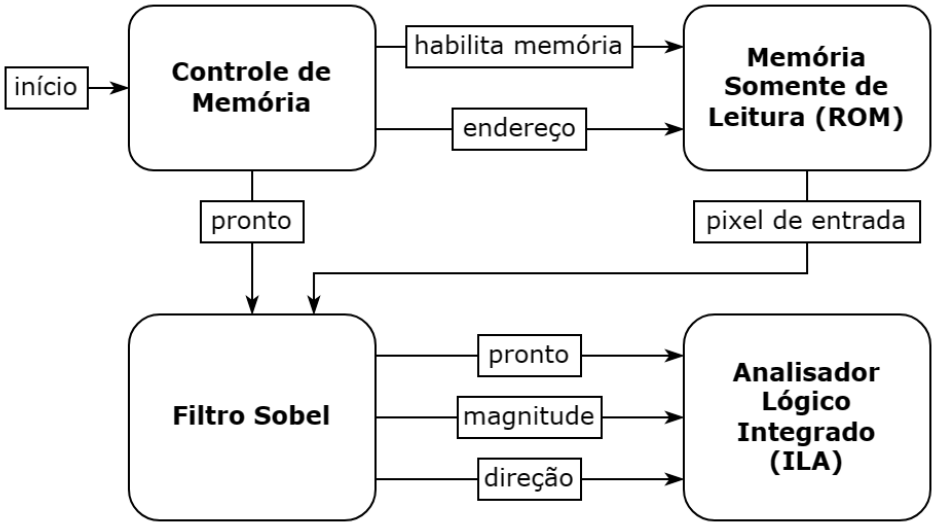


Figura 19 – Visão geral da arquitetura de hardware proposta. Fonte: Elaboração Própria.

3.1 Controle de Memória

A ROM, implementada através de um IPCORE (*Intellectual Property Core*) da Xilinx, é responsável por armazenar os valores dos pixels da imagem de entrada e fornecê-los ao filtro quando requisitada, sendo necessários dois ciclos de relógio para a disponibilização de uma nova saída. O módulo de controle da ROM tem a finalidade de ajustar o fluxo de pixels que vão da memória para o filtro Sobel, além de garantir o atraso de dois ciclos de relógio mencionados. Para isso, foram utilizados dois contadores internos responsáveis pela monitoração do endereço de memória e da passagem dos ciclos de relógio.

Na Figura 19, o sinal de início indica o começo do processo de filtragem como um todo, enquanto que o sinal pronto oriundo do controle de memória sinaliza para o

filtro que uma nova entrada está disponível e pode ser lida, começando a filtragem para um novo pixel. O sinal de endereço obtém seu valor através do contador de endereço de memória, interno ao módulo de controle, realizando a varredura de todos os conteúdos armazenados na ROM através da contagem do primeiro endereço até o último com um intervalo de dois ciclos de relógio. É importante notar que a largura de bits do sinal de endereço depende da resolução da imagem armazenada em memória, já que quanto maior a resolução, maior a quantidade de pixels na imagem e, por consequência, mais endereços devem ser alocados. O sinal que habilita a memória realiza a ativação da ROM para que esta comece a gerar valores para o pixel de entrada.

3.2 Filtro Sobel

O filtro Sobel foi dividido em quatro principais blocos, como descrito na Figura 20. Como as etapas de cálculo do gradiente horizontal e vertical são independentes entre si, estas foram paralelizadas a fim de aumentar a velocidade de execução da filtragem. Entre a etapa de cálculo dos gradientes e o CORDIC, foi feita uma expansão de bits à direita dos sinais Gx e Gy, a fim de considerar a quantidade de bits de precisão necessários para o cálculo em ponto fixo realizado no CORDIC.

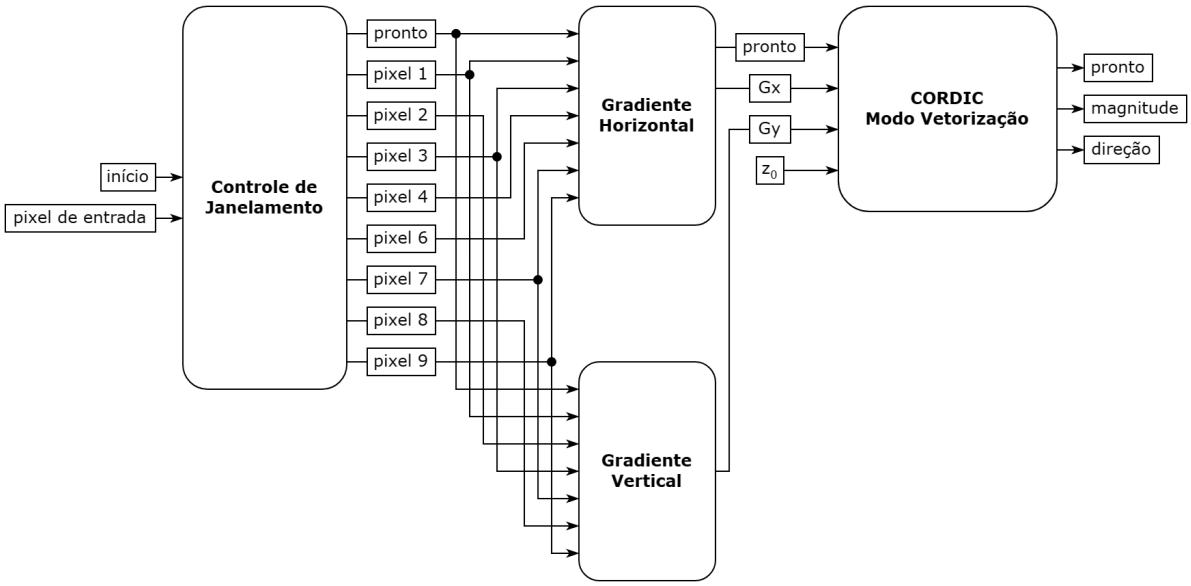


Figura 20 – Arquitetura de hardware do filtro Sobel. Fonte: Elaboração Própria.

3.2.1 Controle de Janelamento

O bloco de controle de janelamento armazena os pixels de entrada vindos da memória em um vetor com esquema FIFO, determinando quando uma janela válida para convolução é formada dentro dessa estrutura, a fim de repassar os valores dos pixels da

janela para os blocos de cálculo dos gradientes horizontal e vertical, conforme mostrado na Figura 20.

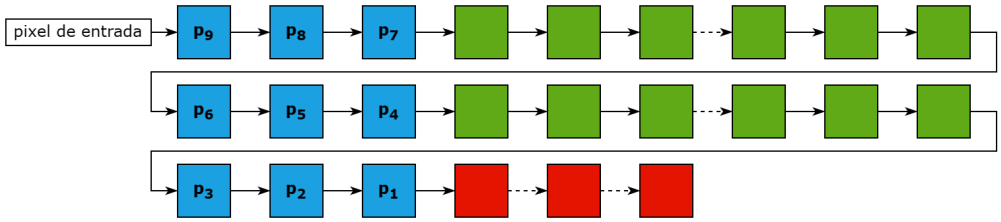


Figura 21 – Vetor FIFO do controle de janelamento. Fonte: Elaboração Própria.

A Figura 21 demonstra a organização do vetor FIFO, formado pelos quadrados azuis e verdes. Os pixels da janela são representados nos quadrados azuis, enquanto que os quadrados verdes são pixels em espera dentro do vetor, que serão utilizados em convoluções posteriores. Os quadrados vermelhos ilustram o fluxo do vetor, mostrando pixels que passaram pelos cálculos necessários e foram descartados da estrutura.

Devido ao fato do *kernel* do filtro Sobel conter três linhas e três colunas, a primeira janela válida só é formada após duas linhas e três pixels da imagem original serem acumuladas no vetor FIFO, com subsequentes janelas válidas sendo formadas a cada novo pixel de entrada que não é sinalizado como um pixel que forma uma janela inválida da imagem. A sinalização desses pixels inválidos é dada através de um contador interno, que conta a quantidade de pixels já passados em uma linha para determinar a chegada dos pixels que se encontram nos limites da imagem.

3.2.2 Gradiente Horizontal e Vertical

Os blocos de gradiente horizontal e vertical são responsáveis pelo cálculo desses respectivos resultados. Devido ao fato desses blocos executarem operações parecidas, ambos entregam suas saídas ao mesmo tempo, o que torna viável a paralelização desses processos. Antes de realizar as operações desses blocos, foi feita a expansão de um bit com todos os pixels da janela de convolução, concatenando um zero à esquerda para considerar o sinal dos valores nas operações seguintes.

A Figura 22 mostra como os pixels presentes no vetor FIFO são distribuídos para os blocos de cálculo dos gradientes. Os *kernels*, representados pelos quadrados cinzas, interagem com os pixels da janela de convolução seguindo uma lógica posicional, no qual a primeira posição da janela de convolução está conectada com a primeira posição de ambos os *kernels*. A multiplicação de cada pixel da janela pelo *kernel* em posição equivalente é feita de forma paralela e esses valores são somados para resultar no gradiente horizontal e vertical do pixel analisado, representados pelos sinais Gx e Gy. Estes sinais de gradiente tem largura de bit equivalente a soma da largura de bit de sinal, do pixel de entrada, e dos *kernels* utilizados.

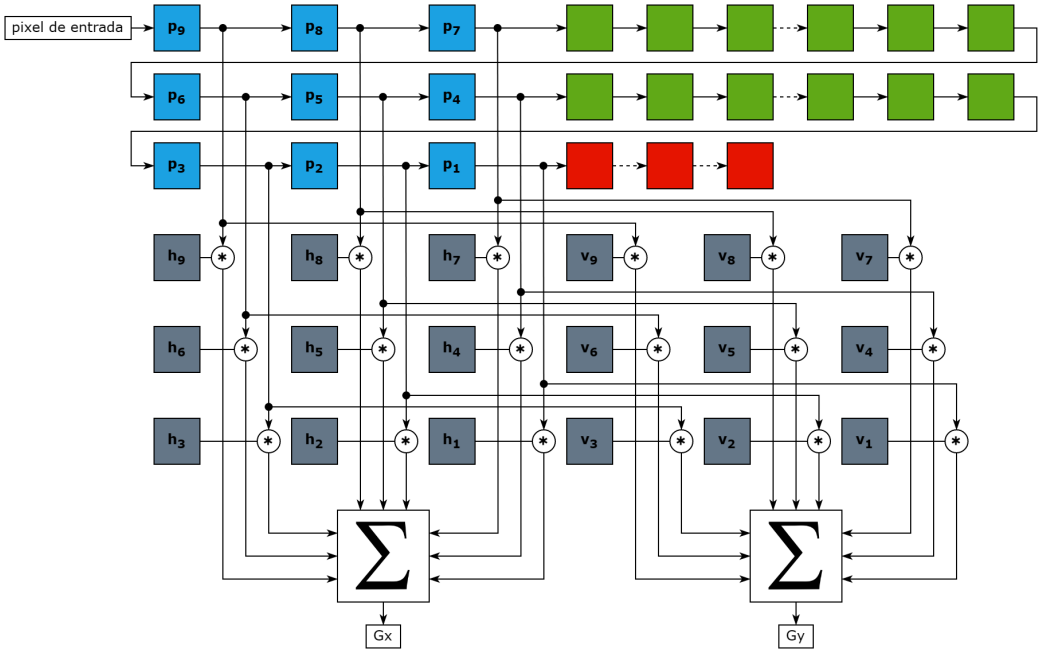


Figura 22 – Integração do controle de janelamento com os blocos de cálculo dos gradientes. Fonte: Elaboração Própria.

É importante ressaltar que a Figura 22 representa um esquema que conecta todos os termos da janela de convolução a fim de mostrar a lógica posicional que pode ser utilizada para qualquer convolução desse tipo. Porém, a implementação com os *kernels* do filtro Sobel permite a omissão de alguns termos que seriam nulos após a multiplicação com o *kernel*, simplificando a estrutura de janelamento para a versão vista na Figura 20, onde alguns termos são utilizados apenas por um dos gradientes, ou nem sequer são utilizados, como é o caso do pixel 5, que representa o pixel central da janela.

3.2.3 CORDIC Modo Vetorização

A última etapa do filtro Sobel utiliza os gradientes horizontal e vertical, previamente calculados, como componentes de coordenada do vetor gradiente, com a finalidade de descobrir a magnitude e direção desse vetor.

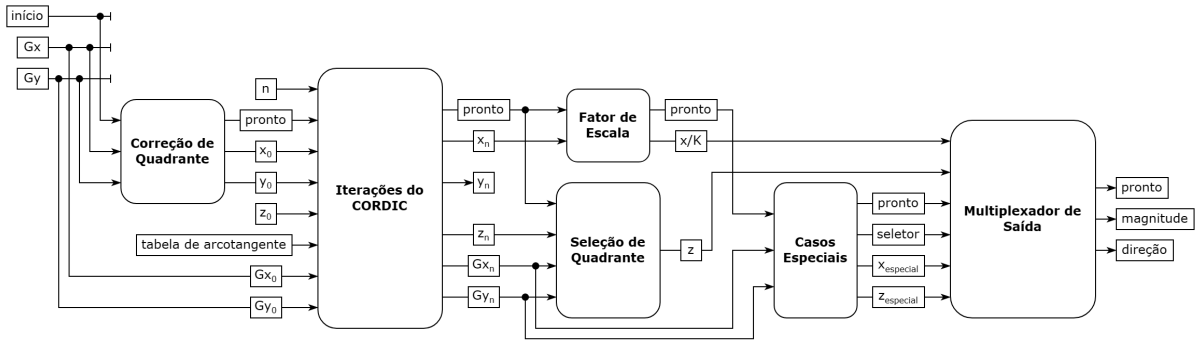


Figura 23 – Arquitetura de hardware do CORDIC. Fonte: Elaboração Própria.

A Figura 23 apresenta a arquitetura de hardware proposta para o CORDIC, separada em seis blocos. O bloco de correção de quadrante retira o módulo dos valores de gradiente horizontal e vertical, deslocando-os para o primeiro quadrante. A finalidade dessa operação é evitar erros nos cálculos realizados nas iterações, já que as equações geradas para o CORDIC tem como base a rotação de um vetor qualquer no domínio do primeiro quadrante, como mostrado na Figura 8.

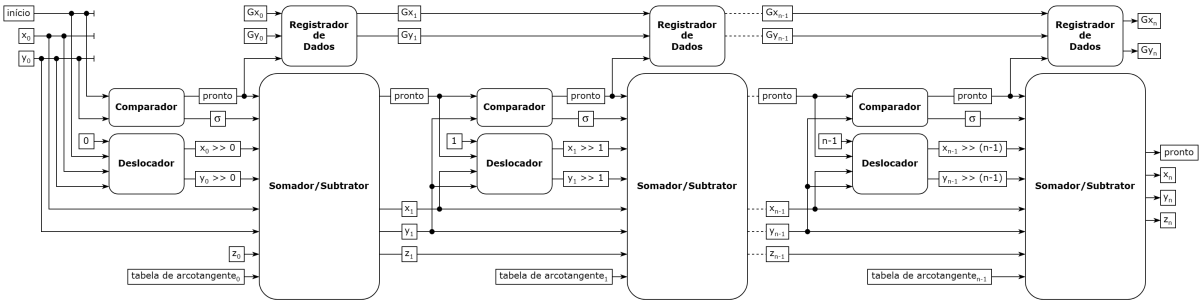


Figura 24 – Visão interna do bloco de iterações do CORDIC. Fonte: Elaboração Própria.

Na Figura 24 são apresentados os componentes internos do bloco de iterações do CORDIC, no qual uma iteração é composta por quatro componentes. Esses componentes são conectados a fim de implementar as Equações (2.20), (2.21) e (2.22), utilizando os parâmetros do modo vetorização. O bloco comparador é responsável por analisar a entrada y, de acordo com a Equação (2.23), enviando um sinal lógico que muda o sinal das operações feitas no bloco somador/subtrator, equivalente e a multiplicação por σ_n. O bloco deslocador realiza o deslocamento à direita das entradas x e y, de acordo com o índice da iteração, implementando a multiplicação pelo termo 2^{-n}. O bloco somador/subtrator implementa a operação final entre a entrada deslocada e a entrada original, adicionando ou subtraindo conforme o valor de σ_n. Por fim, o bloco registrador de dados armazena os valores dos gradientes horizontal e vertical, com a finalidade de propagá-los na estrutura para serem utilizados posteriormente pelos blocos de seleção de quadrante e casos especiais.

Continuando o fluxo da Figura 23, o bloco fator de escala é responsável por retirar o termo K, apresentado previamente na Equação (2.19), que aparece como um fator de escala para a magnitude do gradiente, como apresentado na Equação (2.24). Sendo assim, é necessário multiplicar a saída x_n pelo valor $\frac{1}{K}$ para anular este termo que surge ao final das iterações do CORDIC. Ao mesmo tempo que esse processo ocorre, o bloco seleção de quadrante executa em paralelo, com a finalidade de ajustar o sinal da saída z_n de acordo com o sinal das entradas originais Gx e Gy, tornando a operação de arcotangente descrita na Equação (2.26) válida para os quatro quadrantes. Dessa forma, é feita a conversão do sinal para seu quadrante correto, seguindo as relações mostradas na Figura 25.

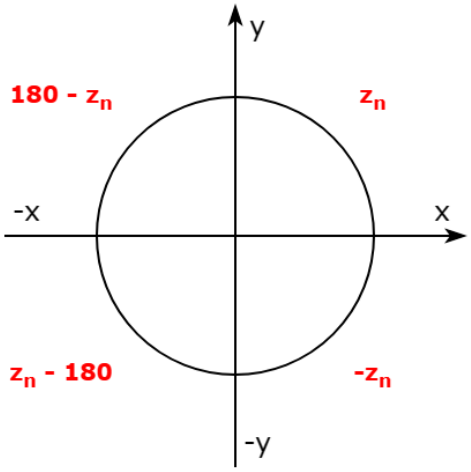


Figura 25 – Relações de conversão de quadrante no ciclo trigonométrico. Fonte: Elaboração Própria.

O bloco casos especiais analisa as entradas originais Gx e Gy para definir a ocorrência de um caso específico que o algoritmo CORDIC não consegue lidar. A Tabela 2 apresenta esses casos, especificando as saídas entregadas pelo bloco de acordo com as entradas. Após essa análise, é enviado ao bloco multiplexador de saída um sinal lógico seletor, que é utilizado para escolher entre as entradas especiais ou as entradas normais, oriundas do cálculo feito pelo algoritmo CORDIC.

Tabela 2 – Entradas e saídas para os casos especiais do CORDIC.

Entrada		**Saída**	
Gx	**Gy**	$\mathbf{x_{especial}}$	$\mathbf{z_{especial}}$
0	0	0	0
0	> 0	$\|Gy\|$	90
0	< 0	$\|Gy\|$	-90
> 0	0	$\|Gx\|$	0
< 0	0	$\|Gx\|$	180

3.3 Analisador Lógico Integrado

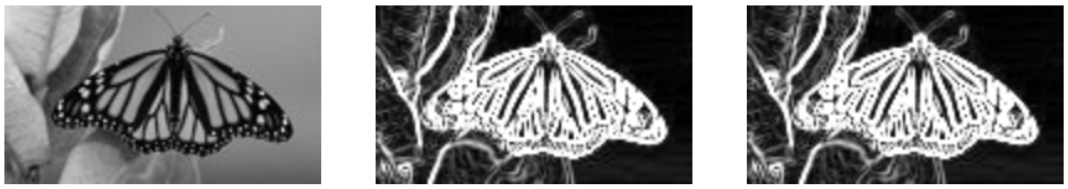
O bloco ILA, que também foi implementado através de um IPCORE da Xilinx, tem a finalidade de monitorar as saídas da arquitetura para realizar a análise pós-implementação no kit de desenvolvimento. Agindo de forma síncrona à arquitetura, os sinais conectados as entradas do analisador são amostrados a partir de um gatilho customizável e armazenados utilizando elementos de memória como BRAMs. Os principais parâmetros que devem ser especificados para esse analisador são a quantidade de entradas, a profundidade da amostra a ser rastreada e a largura de cada entrada (XILINX, 2012).

4 Resultados

Neste capítulo são apresentados os resultados obtidos das simulações comportamentais e da implementação em hardware da arquitetura, além da comparação de performance entre essa implementação e o modelo de referência em software. A arquitetura de hardware proposta foi gerada através da ferramenta *Vivado Design Suite 2018.3*, enquanto o modelo em software foi programado utilizando linguagem C++ e a versão 4.5.5 da biblioteca OpenCV, com as saídas de ambas implementações sendo analisadas através do MATLAB R2021a.

4.1 Simulação Comportamental

Para a simulação comportamental foram utilizadas três imagens com resoluções diferentes, todas em escala de cinza com oito bits de largura e resoluções expandidas através do método de replicação feito previamente no MATLAB, adicionando duas linhas e duas colunas à sua resolução original. O objetivo das simulações foi avaliar visualmente as saídas do filtro, além de medir a latência, taxa de transferência e tempo total de execução para cada resolução.



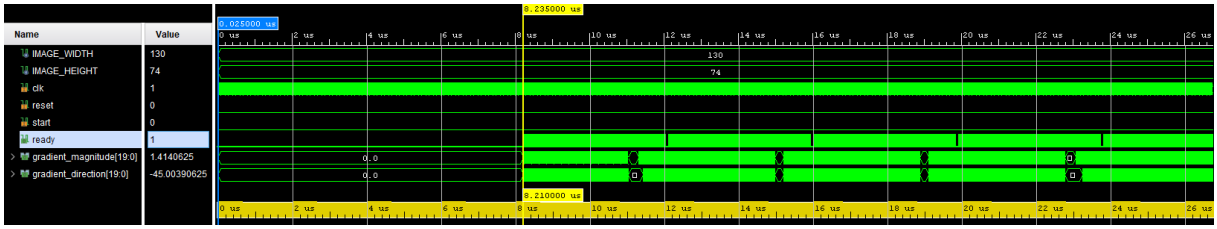
(a) Imagem A - Resolução original de 128x72.

(b) Imagem B - Resolução original de 256x144.

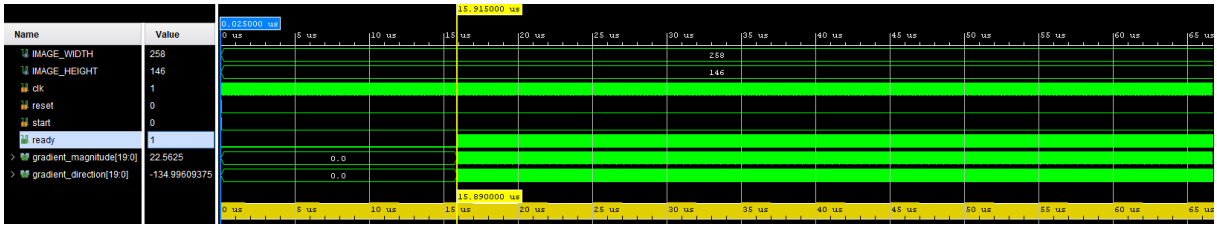
(c) Imagem C - Resolução original de 640x360.

Figura 26 – Comparação entre imagem original e resultados obtidos das filtragens. Fonte: Elaboração Própria.

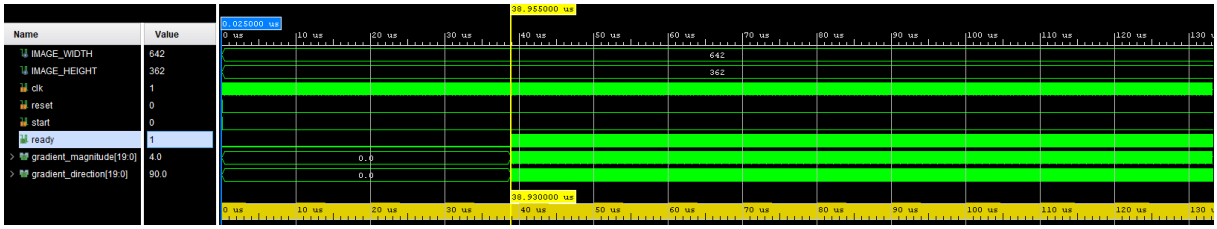
A Figura 26 apresenta a comparação entre a imagem original, na primeira coluna, a filtragem dada pelo modelo de referência em software, na segunda coluna, e a filtragem dada pela arquitetura em hardware, representada na terceira coluna. Percebe-se que, visualmente, as imagens filtradas são semelhantes, o que é melhor elaborado através da discussão sobre o erro quadrático médio apresentada no tópico de implementação em hardware.



(a) Imagem A - Latência de 8,21 us.



(b) Imagem B - Latência de 15,89 us.



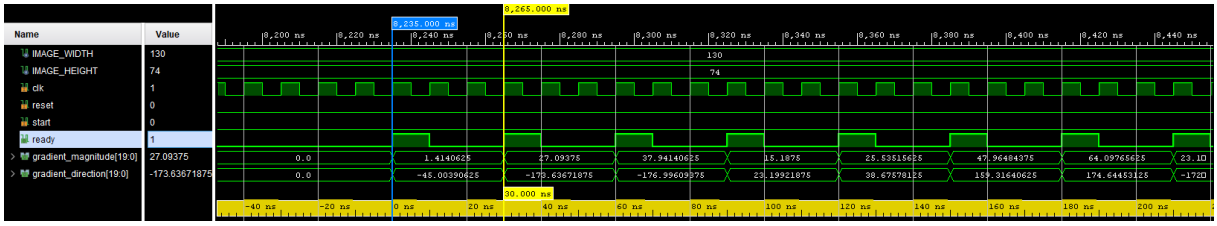
(c) Imagem C - Latência de 38,93 us.

Figura 27 – Latência da arquitetura proposta. Fonte: Elaboração Própria.

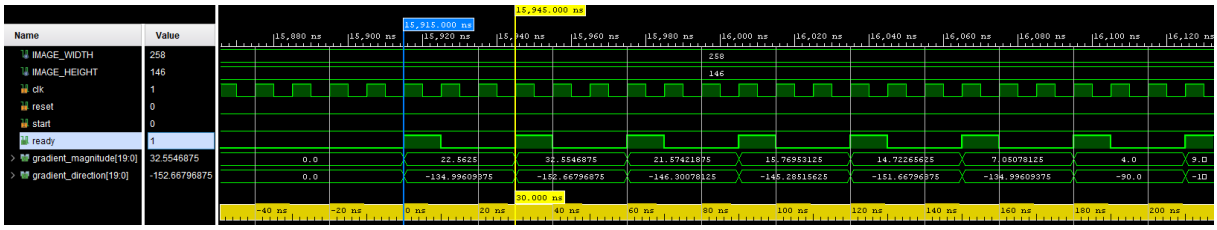
As latências apresentadas na Figura 27 foram medidas através da diferença de tempo entre a ativação do sinal de início e a primeira ativação do sinal pronto da arquitetura. Ao comparar as latências, percebe-se a influência da resolução da imagem sobre esse resultado, já que o vetor no esquema FIFO responsável pelo janelamento necessita acumular duas linhas e três pixels da imagem de entrada para começar a calcular o gradiente. Descrevendo a latência da arquitetura como a soma das latências dos blocos menores, com frequência de relógio f, temos que $L = (L_{FIFO} + L_{GRAD} + L_{CORDIC}) * \frac{1}{f}$. Considerando uma imagem de resolução MxN qualquer, cujas bordas serão expandidas pelo método de replicação, o termo L_{FIFO} é descrito pelo acúmulo de duas linhas e três pixels da imagem de entrada, sendo que cada pixel leva três ciclos de relógio para entrar na arquitetura, com mais um ciclo necessário para que a arquitetura acuse o preenchimento do vetor FIFO e prossiga com o processo, resultando em $L_{FIFO} = (((M + 2) * 2) + 3) * 3) + 1$. O termo L_{GRAD} equivale a dois ciclos de relógio, um para as multiplicações dos pixels da janela pelo *kernel* do filtro e outro para a soma desses valores. Por fim, o termo L_{CORDIC}

equivale a 29 ciclos de relógio, sendo um ciclo para a correção do quadrante, dois ciclos para cada uma das 12 iterações do CORDIC, dois ciclos para o fator de escala/seleção de quadrante, um ciclo para os casos especiais e mais um ciclo para o multiplexador de saída. Dessa forma, a latência da arquitetura em função da resolução da imagem é dada pela Equação (4.1).

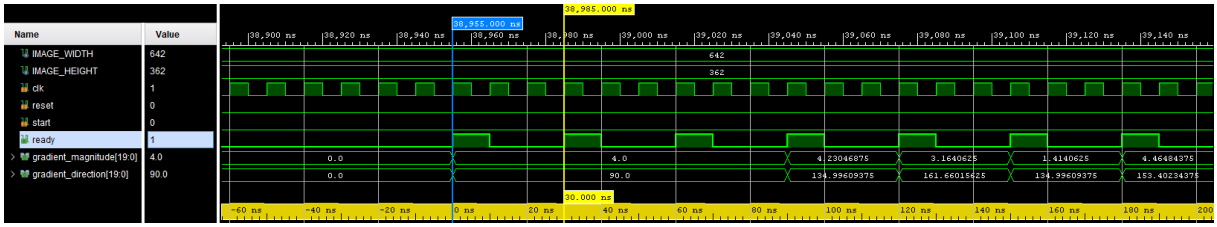
$$L = \frac{((((M+2)*2)+3)*3)+1+2+29)}{f} \tag{4.1}$$



(a) Imagem A - Período de 30 ns entre saídas.



(b) Imagem B - Período de 30 ns entre saídas.



(c) Imagem C - Período de 30 ns entre saídas.

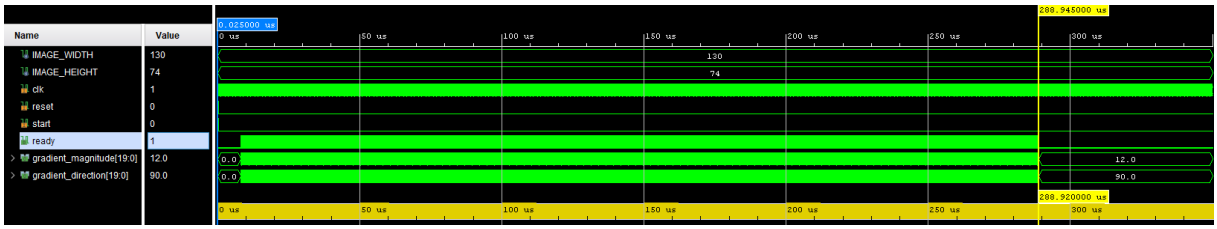
Figura 28 – Período entre saídas da arquitetura proposta. Fonte: Elaboração Própria.

Na Figura 28 é apresentado o período entre saídas para filtragem das imagens de teste, que foi medido através da diferença de tempo entre a ativação de dois sinais pronto da arquitetura. A taxa de transferência (ou *throughput*) consiste no inverso desse período, resultando em 33 MP/s. Comparando os resultados mostrados, percebe-se que a taxa de transferência não é afetada pela resolução da imagem. Isso se deve ao fato da imagem de entrada ter sido organizada em forma de vetor linha na ROM, além da organização em *pipeline* da arquitetura, que permite o enfileiramento das entradas na arquitetura para o processamento em sequência das saídas. Dessa forma, a taxa de transferência é afetada apenas pela frequência do sistema, sendo expressa pela Equação 4.2.

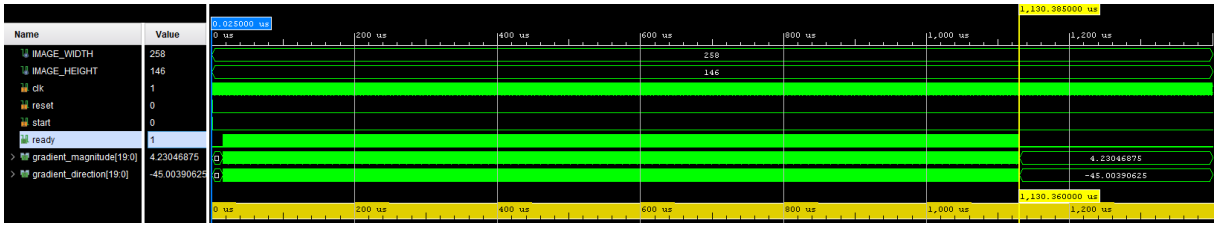
$$T = \frac{f}{3} \tag{4.2}$$

Por fim, a Figura 29 mostra o tempo total de execução para filtragem de cada uma das imagens de teste. Esses valores foram medidos através da diferença de tempo entre a ativação do sinal início e a última ativação do sinal pronto da arquitetura. Dessa forma, o tempo total de execução da arquitetura é expresso em função da latência, da taxa de transferência e de uma resolução MxN qualquer que será expandida utilizando o método de replicação, resultando na Equação (4.3).

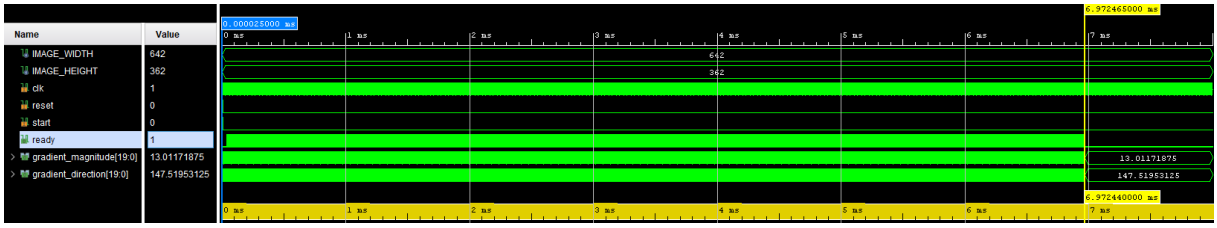
$$t_{total} = L + \frac{((M+2)*(N+2)) - (((M+2)*2)+3)}{T} \tag{4.3}$$



(a) Imagem A - Tempo de execução de 0,28 ms.



(b) Imagem B - Tempo de execução de 1,13 ms.



(c) Imagem C - Tempo de execução de 6,97 ms.

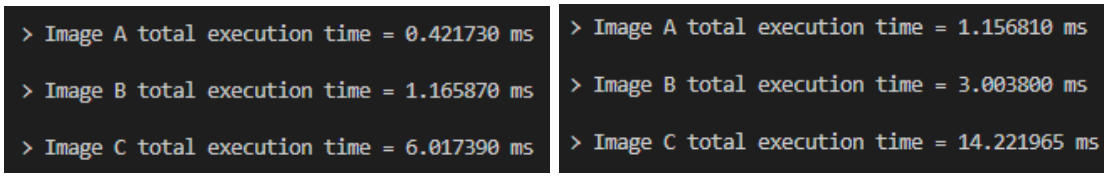
Figura 29 – Tempo total de execução da arquitetura proposta. Fonte: Elaboração Própria.

4.2 Comparação do Tempo de Execução Entre Modelos

O modelo de referência em software foi executado em dois computadores distintos, a fim de levar em consideração o impacto que os componentes do sistema tem sobre o tempo de execução do algoritmo.

Tabela 3 – Especificações gerais dos computadores utilizados.

Computador	**CPU**	**GPU**
A	AMD Ryzen 5 3600X 3.8 GHz	AMD Radeon RX 6800 16 GB
B	AMD Ryzen 7 3700U 2.3 GHz	AMD Radeon RX Vega 2 GB



(a) Tempo de execução no computador A. (b) Tempo de execução no computador B.

Figura 30 – Média de vinte execuções do modelo de referência em dois computadores diferentes. Fonte: Elaboração Própria.

Levando em consideração as especificações da Tabela 3 e observando os resultados obtidos na Figura 30, nota-se que o computador A executa mais rapidamente o algoritmo. Isso se deve ao fato do computador A ter mais capacidade de processamento que o computador B, com uma maior frequência base de CPU e uma GPU moderna com mais memória, o que afeta o algoritmo do filtro Sobel em específico devido à aceleração em hardware que é utilizada pela biblioteca OpenCV para o processamento de imagens.

Tabela 4 – Comparação do tempo total de execução entre sistemas.

Sistema	**Tempo Total de Execução (ms)**			**Frequência (MHz)**
	Imagem A	**Imagem B**	**Imagem C**	
Hardware	0,2889	1,1303	6,9724	100
Computador A	0,4217	1,1658	6,0173	3800
Computador B	1,1568	3,0038	14,2219	2300

A Tabela 4 mostra a comparação entre a velocidade da arquitetura de hardware e o modelo de referência em software executado nos dois computadores. Observando os resultados, é possível notar que a execução da arquitetura de hardware foi mais rápida que o computador 2 em todas as imagens, porém foi mais lenta que o computador 1 apenas na imagem C, que é de maior resolução. Essa maior velocidade de execução da arquitetura de hardware foi alcançada utilizando uma frequência de operação bem menor que a dos computadores, o que demonstra o valor de um dispositivo especializado nessas aplicações.

4.3 Implementação em Hardware

Na implementação em hardware foi utilizada apenas a menor imagem, com resolução de 128x72, devido à limitação de elementos BRAM na placa escolhida, o que impede a implementação da memória e do analisador lógico. A implementação da arquitetura foi realizada com o objetivo de avaliar o uso de recursos de hardware, estimar o consumo de energia e observar a área ocupada pela arquitetura na placa, além de avaliar a exatidão dos resultados em comparação com o modelo de referência através do cálculo do erro quadrático médio. Foram utilizadas três bits de largura para representação do *kernel* do filtro

Sobel e oito bits de largura como precisão do ponto fixo para as entradas do algoritmo CORDIC, totalizando 20 bits de largura para a saída do filtro.

Tabela 5 – Recursos consumidos pela arquitetura de hardware.

Módulo	***LUT***	***FF***	***BRAM***	***DSP***	***I/O Pin***
Topo	2441	3832	40,5	1	3
Controle de Memória	36	32	0	0	0
ROM	14	10	2,5	0	0
Filtro Sobel	1049	1712	0	1	0
ILA	1342	2078	38	0	0

A Tabela 5 apresenta os recursos consumidos pela arquitetura de hardware proposta neste trabalho, com o módulo topo sendo o módulo que abriga todos os outros. O consumo de *LUTs* é explicado através da quantidade de operações de soma, subtração e deslocamento presentes na lógica, enquanto que o consumo de *FFs* é dado pela necessidade de diversos registradores na arquitetura. Percebe-se que o módulo do filtro Sobel acumulou por si só grande parte do uso de *LUTs* e *FFs*, o que pode ser explicado pelo fato desse módulo realizar a maioria das operações aritméticas da arquitetura, além de abrigar o vetor FIFO do controle de janelamento. A utilização de *BRAMs* foi dada pelo módulo ROM e o ILA, como esperado, já que estes módulos fazem uso direto de elementos de memória para armazenamento de valores. Apenas um *DSP* foi utilizado, sendo necessário na multiplicação realizada no bloco fator de escala dentro do algoritmo CORDIC presente no filtro Sobel. Considerando a quantidade de entradas e saídas presentes na arquitetura geral, nota-se que o consumo de pinos de *I/O* foi condizente, já que foram utilizados apenas três sinais de entrada no módulo topo da arquitetura.

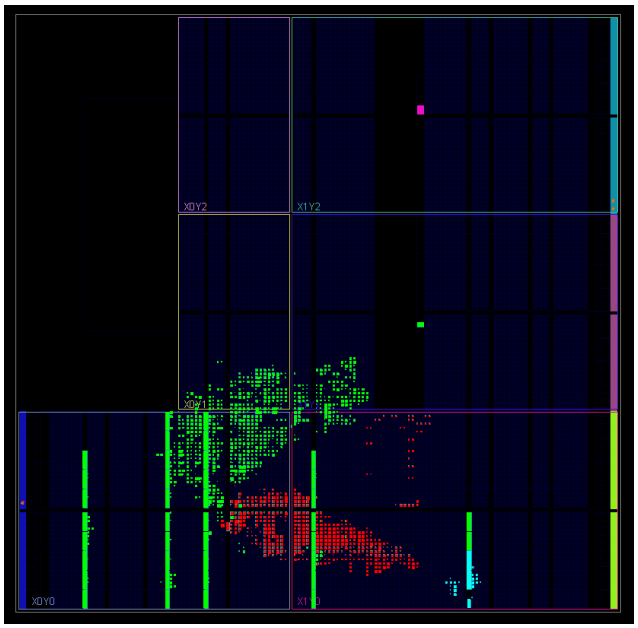


Figura 31 – Layout da arquitetura na placa. Fonte: Elaboração Própria.

A Figura 31 mostra o espaço ocupado pela arquitetura de hardware na placa em que foi implementada. Os blocos azul claro representam a memória e seu módulo de controle, enquanto que os blocos vermelho e verde representam, respectivamente, o filtro Sobel e o analisador lógico. Observando a figura percebe-se que a área ocupada pelos módulos foi condizente com o consumo de hardware descrito na Tabela 5, com o analisador lógico sendo o módulo de maior tamanho da arquitetura, seguido do filtro Sobel.

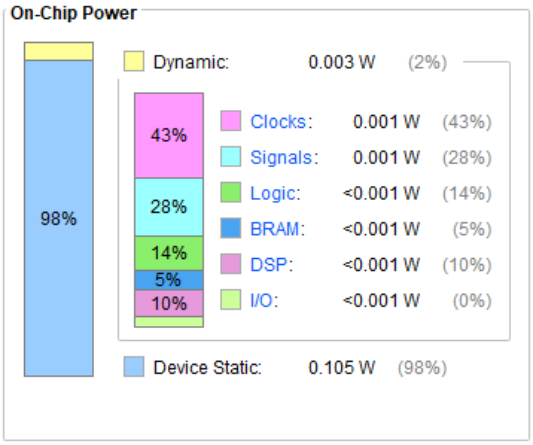


Figura 32 – Estimativa do consumo de energia da arquitetura. Fonte: Elaboração Própria.

Uma estimativa do consumo de energia da arquitetura é apresentado na Figura 32, com um consumo total de 108 mW. Analisando a figura, nota-se que esse gasto energético é dado majoritariamente pelos sinais de relógio que ditam a frequência de funcionamento dos módulos, mas também há contribuições significativas dos sinais lógicos presentes na arquitetura. O único *DSP* utilizado contribui com 10% da energia dinâmica consumida, o que demonstra a complexidade e custo desse elemento.

Por fim, foi feita uma avaliação da exatidão dos resultados da arquitetura de hardware através do cálculo do erro quadrático médio. Esta técnica é utilizada para verificar a acúracia de modelos, dando maior peso para erros maiores, já que cada erro é elevado ao quadrado individualmente antes do cálculo da média em si. Comparando as matrizes de magnitude e direção do gradiente vindas do modelo de referência e da arquitetura de hardware, foi obtido um erro quadrático médio de $4,0125 \cdot 10^{-1}$ para a magnitude e $4,1938 \cdot 10^{-4}$ para a direção do gradiente, demonstrando a exatidão da arquitetura proposta e explicando a semelhança visual das imagens da Figura 26.

5 Conclusão

A detecção de objetos em tempo real é uma tecnologia com inúmeras aplicações, que tende a ser cada vez mais utilizada em diversas áreas com o avanço humano. Sendo a definição de recursos como as bordas de uma imagem uma etapa onerosa destes algoritmos, torna-se necessário acelerá-los, o que pode ser feito através da implementação de uma arquitetura de hardware do filtro Sobel. A proposta de uma arquitetura que aproveite ao máximo a capacidade de paralelizar operações, garantindo uma boa exatidão dos resultados, torna-se imprescindível na busca por uma melhor eficiência nestes algoritmos.

A arquitetura descrita neste trabalho utilizou o esquema de *pipeline* com a finalidade de executar simultâneamente a maior quantidade possível de operações. Cálculos complexos foram destrinchados e simplificados em procedimentos mais simples, utilizando precisão de ponto fixo, em uma tentativa de atingir um equilíbrio entre velocidade de execução e exatidão dos resultados. Além disso, foi parametrizada a resolução da imagem de entrada, tornando possível a adaptação da arquitetura para qualquer outra resolução. Vale ressaltar também que, como a ideia principal da arquitetura se baseia na convolução de um *kernel* 3x3 com uma imagem qualquer, esta pode ser utilizada com outros filtros baseados em convolução, necessitando apenas de algumas modificações no código.

Analisando os resultados exibidos no capítulo anterior, pode-se dizer que a arquitetura cumpriu o seu propósito. No geral, o tempo de execução da arquitetura de hardware foi menor que um computador de mesa com recursos mais potentes, mostrando a vantagem que um circuito dedicado tem em relação a uma plataforma genérica. Levando em conta as taxas de atualização de vídeo mais comuns, de 30 à 60 quadros por segundo, o tempo de execução se encaixou nos parâmetros de tempo real exigido por muitas aplicações. Em relação a exatidão das saídas de magnitude e direção do gradiente, foi comprovado tanto visualmente quanto numericamente que os resultados foram satisfatórios, com um erro quadrático médio baixo e uma comparação visual extremamente semelhante. Dessa forma, os resultados alcançados neste trabalho refletem uma solução embarcada do filtro Sobel que atende os requisitos de tempo real e possibilita sua aplicação em plataformas móveis com restrições de tamanho, peso e consumo energético.

Referências

BABICH, N. *What Is Computer Vision & How Does it Work? An Introduction*. 2020. Disponível em: <https://xd.adobe.com/ideas/principles/emerging-technology/what-is-computer-vision-how-does-it-work/>. Acesso em: 18 de set. de 2022. Citado na página 15.

CHAPLE, G.; DARUWALA, R. D. Design of sobel operator based image edge detection algorithm on fpga. 2014. Citado 4 vezes nas páginas 9, 29, 30 e 33.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. 2005. Citado 3 vezes nas páginas 9, 15 e 16.

DASIOPOULOU, S. et al. Knowledge-assisted semantic video object detection. 2005. Citado na página 15.

EB. *XILINX ARTIX ARTY*. 2015. Disponível em: <https://eb.dy.fi/2015/11/xilinx-artix-arty/>. Acesso em: 19 de set. de 2022. Citado 2 vezes nas páginas 9 e 26.

ERCEGOVAC, M. D.; LANG, T. *Digital Arithmetic*. 1st. ed. [S.l.]: Elsevier, 2003. Citado 4 vezes nas páginas 9, 23, 24 e 25.

FRITZ. *Object Detection Guide*. 2021. Disponível em: <https://www.fritz.ai/object-detection/>. Acesso em: 18 de set. de 2022. Citado na página 15.

GHAFFARI, S. et al. Fpga-based implementation of hog algorithm: Techniques and challenges. 2019. Citado na página 17.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014. Citado na página 15.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 4th. ed. [S.l.]: Pearson Education, 2018. Citado 3 vezes nas páginas 19, 20 e 21.

GUO, Z.; XU, W.; CHAI, Z. Image edge detection based on fpga. 2010. Citado 3 vezes nas páginas 9, 31 e 33.

IBM. *What is computer vision?* s.d. Disponível em: <https://www.ibm.com/topics/computer-vision>. Acesso em: 18 de set. de 2022. Citado na página 15.

INTEL. *What is an SoC FPGA?* 2014. Disponível em: <https://www.intel.com/content/dam/support/us/en/programmable/support-resources/bulk-container/pdfs/literature/ab/ab1-soc-fpga.pdf>. Acesso em: 19 de set. de 2022. Citado na página 27.

JACOBS, D. Image gradients. 2005. Citado na página 19.

MARR, B. *7 Amazing Examples Of Computer And Machine Vision In Practice*. 2019. Disponível em: <https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/?sh=47513d51018c>. Acesso em: 18 de set. de 2022. Citado na página 15.

MUNOZ, D. M. et al. Low latency disturbance detection using distributed optical fiber sensors. 2017. Citado 3 vezes nas páginas 9, 32 e 33.

NAUSHEEN, N. et al. A fpga based implementation of sobel edge detection. 2018. Citado 4 vezes nas páginas 9, 28, 29 e 33.

SOBEL, I. An isotropic 3x3 image gradient operator. 2014. Citado na página 20.

VINCENT, O. R.; FOLORUNSO, O. A descriptive algorithm for sobel image edge detection. 2009. Citado na página 19.

VOLDER, J. E. The cordic trigonometric computing technique. 1959. Citado na página 23.

XILINX. *Integrated Logic Analyzer v2.0 Data Sheet (DS875)*. 2012. Disponível em: <https://docs.xilinx.com/v/u/en-US/ds875-ila>. Acesso em: 30 de mar. de 2023. Citado na página 39.

XILINX. *What is an FPGA?* 2022. Disponível em: <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>. Acesso em: 19 de set. de 2022. Citado na página 26.

XILINX. *Zynq-7000 SoC Product Advantages*. 2022. Disponível em: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#productAdvantages>. Acesso em: 19 de set. de 2022. Citado 2 vezes nas páginas 9 e 27.