Trabalho de Conclusão de Curso

Processamento Colaborativo ARM-FPGA no Cyclone V SoC: Estudo Comparativo de Arquiteturas com Cortex-A9 e Raspberry Pi 5 para Reconhecimento de Gestos

de **Pedro Henrique de Brito Nascimento**

orientado por
Prof. Dr. Ícaro Bezerra Queiroz de Araújo
M.e Gustavo Costa Gomes de Melo

Universidade Federal de Alagoas
Instituto de Computação
Maceió, Alagoas
5 de Dezembro de 2024

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

PROCESSAMENTO COLABORATIVO ARM-FPGA NO CYCLONE V SOC: ESTUDO COMPARATIVO DE ARQUITETURAS COM CORTEX-A9 E RASPBERRY PI 5 PARA RECONHECIMENTO DE GESTOS

Trabalho de Conclusão de Curso submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para a obtenção do grau de Engenheiro de Computação.

Pedro Henrique de Brito Nascimento

Orientador: Prof. Dr. Ícaro Bezerra Queiroz de Araújo
Coorientador: M.e Gustavo Costa Gomes de Melo

Banca Avaliadora:

Prof. Dr. Allan de Medeiros Martins UFRN
Prof. Dr. João Raphael Souza Martins UFAL

Maceió, Alagoas
5 de Dezembro de 2024

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecária Responsável: Lívia Silva dos Santos - CRB 1670

N244p Nascimento, Pedro Henrique de Brito.
Processamento colaborativo ARM-FPGA no cyclone V SoC : estudo comparativo de arquiteturas com córtex-A9 e raspberry Pi 5 para reconhecimento de gestos / Pedro Henrique de Brito Nascimento. –2024
72 f.:il. color.

Orientador: Ícaro Bezerra Queiroz de Araújo.
Coorientador: Gustavo Costa de Melo.
Monografia (Trabalho de Conclusão de Curso em Engenharia da Computação) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2024.

Bibliografia: f. 70-72

1. Arquitetura híbrida de HPS-FPGA. 2. Reconhecimento de gestos. 3. Sistemas embarcados (computação). I. Título.

CDU: 004.75:004.451

Agradecimentos

Agradeço a Deus por me guiar, fortalecer e iluminar meus passos em cada etapa desse caminho. Aos meus pais, Maria Cristina e Paulo, minha base em tudo, agradeço por cada sacrifício, cada palavra de apoio e por nunca deixarem de acreditar em mim. Tudo o que conquistei até aqui é resultado da força e dos valores que recebi de vocês. Sou profundamente grato por todo o amor, compreensão e presença incondicional ao longo dessa jornada. Ao meu irmão, Luccas, agradeço por estar ao meu lado nos momentos em que mais precisei. Nossas conversas sobre a vida e o futuro me ajudaram a encontrar clareza quando tudo parecia incerto. À minha namorada, Lilian, agradeço por estar comigo em cada etapa dessa jornada. Vivemos juntos os altos e baixos da graduação, enfrentamos dificuldades, comemoramos vitórias e aprendemos muito um com o outro no caminho. Seu apoio, compreensão e companheirismo foram fundamentais para que eu conseguisse chegar até aqui.

Aos professores e orientadores que caminharam comigo nesse processo, agradeço especialmente ao professor Ícaro, cuja orientação foi marcada por objetividade, compromisso e sensibilidade. Sou grato pela confiança, pela escuta atenta e por ter me guiado com clareza em cada etapa. Ao Gustavo, meu co-orientador, agradeço pelas contribuições técnicas fundamentais e por estar sempre disponível quando precisei de direção. Aos professores Allan e João Raphael, pela leitura atenta, pelas observações valiosas e por terem enriquecido este trabalho com seu olhar crítico e construtivo. Aos colegas de curso, com quem compartilhei desafios, aprendizados e conquistas, meu muito obrigado pela parceria e pelas experiências que levarei para a vida. Agradeço também à Universidade Federal de Alagoas, que me proporcionou acesso ao conhecimento e à construção da minha formação. Ao Centro de Inovações (*EDGE*) e ao Centro de Pesquisa em Engenharia e Sistemas *(EASY*), por abrirem portas, oferecerem oportunidades e contribuírem diretamente para minha evolução acadêmica e profissional.

Por fim, agradeço a todos — amigos, conhecidos, professores e profissionais — que, direta ou indiretamente, contribuíram para que este trabalho fosse possível. Cada palavra de incentivo, cada gesto de apoio e cada troca de experiência deixaram uma marca neste percurso que agora se encerra.

Muito obrigado.

Resumo

Sistemas embarcados desempenham um papel central em diversas aplicações que exigem processamento em tempo real, combinando hardware e software para executar funções específicas. Dentro desse contexto, as arquiteturas híbridas, que integram diferentes tecnologias, como microcontroladores, FPGAs e SoCs, destacam-se por sua capacidade de aliar flexibilidade de processamento e eficiência energética. Um exemplo de tais tecnologias é o kit de desenvolvimento DE1-SoC, que combina um processador ARM Cortex-A9 com uma FPGA Cyclone V. Este trabalho apresenta uma análise comparativa de arquiteturas híbridas de sistemas embarcados, aplicadas ao processamento digital de imagens (PDI), utilizando o reconhecimento de gestos como estudo de caso. Foram avaliadas três configurações principais: uma abordagem baseada exclusivamente na Raspberry Pi 5, uma arquitetura colaborativa entre a Raspberry Pi 5 e a FPGA Cyclone V, e uma solução totalmente integrada utilizando o DE1-SoC. O objetivo central foi investigar as vantagens e limitações de cada configuração em termos de tempo de resposta, eficiência no processamento e escalabilidade. Os resultados destacaram que a arquitetura colaborativa entre a Raspberry Pi 5 e o FPGA apresentou um desempenho superior, com uma redução de aproximadamente 89,1% no tempo total de execução em relação a Raspberry Pi 5 utilizada de forma independente. Em contrapartida, a solução integrada ao Cortex-A9 demonstrou um tempo de execução cerca de 29,8 vezes maior na transferência de dados, evidenciando gargalos em operações de entrada e saída. A Raspberry Pi 5 isolada, por sua vez, mostrou-se insuficiente para demandas em tempo real, destacando suas limitações em cenários com *deadlines* estritos. Com base nesses resultados, o trabalho reforça a importância do planejamento arquitetural em sistemas embarcados e sugere futuras pesquisas voltadas à otimização de protocolos de comunicação e melhorias no gerenciamento de sistemas operacionais para aplicações de alto desempenho.

***Palavras-chave*: *FPGA*; *HPS*; *DE1-SOC*; Reconhecimento de Gestos; Processamento Colaborativo; Arquitetura Híbrida de HPS-FPGA.**

Abstract

Embedded systems play a central role in various applications that require real-time processing, combining hardware and software to perform specific functions. In this context, hybrid architectures, which integrate different technologies such as microcontrollers, FPGAs, and SoCs, stand out for their ability to combine processing flexibility and energy efficiency. An example of such architectures is the DE1-SoC development kit, which combines an ARM Cortex-A9 processor with a Cyclone V FPGA. This work presents a comparative analysis of hybrid embedded system architectures applied to digital image processing (DIP), using gesture recognition as a case study. Three main configurations were evaluated: an approach based exclusively on the Raspberry Pi 5, a collaborative architecture between the Raspberry Pi 5 and the Cyclone V FPGA, and a fully integrated solution using the DE1-SoC. The central objective was to investigate the advantages and limitations of each configuration in terms of response time, processing efficiency, and scalability. The results highlighted that the collaborative architecture between the Raspberry Pi 5 and the FPGA achieved superior performance, with a reduction of approximately 89.1% in total execution time compared to the Raspberry Pi 5 used independently. In contrast, the integrated solution with the Cortex-A9 showed an execution time about 29.8 times greater in data transfer, revealing bottlenecks in input and output operations. The isolated Raspberry Pi 5, in turn, proved insufficient for real-time demands, highlighting its limitations in scenarios with strict deadlines. Based on these results, the work emphasizes the importance of architectural planning in embedded systems and suggests future research focused on optimizing communication protocols and improving operating system management for high-performance applications.

***Keywords*: *FPGA*; *HPS*; *DE1-SOC*; *Gesture Recognition*; Collaborative Processing; HPS-FPGA Hybrid Architecture.**

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ALM *Adaptive Logic Module*

ARM *Advanced RISC Machine*

AXI *Advanced eXtensible Interface*

DHCP *Dynamic Host Configuration Protocol*

DNS *Domain Name System*

DSP *Digital Signal Processor*

EDS *Embedded Development System*

FPGA *Field Programmable Gate Array*

GCC *GNU Compiler Collection*

GND *Ground Reference*

GPIO *General-Purpose Input/Output*

HPS *Hard Processor System*

I2C *Inter-Integrated Circuit*

IDE *Integrated Development Environment*

LAN *Local Area Network*

LUT *Lookup Table*

MISO *Master In Slave Out*

MSB *Most Significant Bit*

MOSI *Master Out Slave In*

PDI Processamento Digital de Imagem

PLL *Phase-Locked Loop*

RGB Espaço de cores Vermelho, Verde, Azul

RTOS *Real Time Operating System*

RTL *Register Transfer Level*

SCK *Serial Clock*

SDHC *Secure Digital High Capacity*

SOC *System On-Chip*

SPI *Serial Peripheral Interface*

SSH *Secure Shell*

SS *Slave Select*

USB *Universal Serial Bus*

VLSI *Very-large-scale integration*

WAN *Wide-Area Network*

YCbCr Espaço de cores Luminância, Crominância Azul, Crominância Vermelha

Sumário

Capítulo 1

Introdução

Muito se tem discutido recentemente acerca da importância dos sistemas embarcados híbridos, que integram componentes programáveis, como FPGAs, e processadores de propósito geral. Essa combinação possibilita avanços em áreas que demandam alto desempenho, como processamento de sinais, visão computacional e aprendizado de máquina. Projetados para aplicações de alta complexidade, esses sistemas destacam-se pela flexibilidade das arquiteturas híbridas, eficiência energética e capacidade de paralelismo dos FPGAs, fatores que tornam a área fundamental para o desenvolvimento de soluções inovadoras em setores como automação, saúde e transporte autônomo.

Estudos recentes mostram que arquiteturas que mesclam processadores integrados HPS (*Hard Processor System*) — sistemas baseados em processadores dedicados implementados diretamente no hardware, integrados com outros elementos como memórias e interfaces de comunicação — e *FPGA (Field Programmable Gate Array)* apresentam melhorias em desempenho e eficiência energética quando comparadas a soluções baseadas exclusivamente em processadores tradicionais, conforme indicado em Carmonal et al. (2021) e Paratane et al. (2016). Aplicações críticas, como monitoramento cardíaco e direção autônoma, destacam-se como exemplos práticos dessa abordagem, discutidas por Hu et al. (2024) e Yun and Park (2024). O uso de linguagens de alto nível, como *OpenCL*, facilita o desenvolvimento de soluções avançadas em plataformas que exigem otimização de tarefas intensivas em tempo real, destacam Bouhoun et al. (2020). No entanto, ainda existem desafios nesse contexto, como a necessidade de otimizar a comunicação entre componentes, reduzir a complexidade do design de hardware e software e desenvolver algoritmos que mantenham um equilíbrio eficaz entre desempenho e consumo energético, conforme observado em Yun and Park (2024).

Frente aos desafios e às possibilidades da integração HPS e FPGA, o presente trabalho tem como objetivo investigar e comparar arquiteturas de processamento colaborativo, analisando seu desempenho em relação a abordagens baseadas em processamento individualizado. O estudo de caso escolhido é o reconhecimento de gestos, utilizado para avaliar as vantagens e limitações de cada abordagem. O trabalho visa contribuir para o

desenvolvimento de soluções eficientes e escaláveis, adequadas a cenários que exigem alta complexidade, processamento intensivo e baixa latência.

1.1 Objetivos

1.1.1 Objetivo geral

Este trabalho visa desenvolver e comparar diferentes metodologias para a implementação de um sistema de reconhecimento de gestos, utilizando soluções baseadas em hardware dedicado e sistemas embarcados. O intuito é avaliar essas abordagens em termos de desempenho e escalabilidade.

1.1.2 Objetivos específicos

- Implementar e avaliar uma arquitetura de processamento colaborativo que utiliza a Raspberry Pi 5 para gerenciar a comunicação e o envio de imagens, enquanto delega o processamento de imagens e a classificação para a FPGA Cyclone V do DE1-SoC, utilizando o protocolo SPI para integração.
- Implementar a mesma arquitetura de processamento colaborativo, mas totalmente integrada no DE1-SoC, utilizando o ARM Cortex-A9 para controle da comunicação e envio de imagens, e o FPGA Cyclone V, também interligados via SPI. Avaliar o desempenho desta configuração.
- Implementar e avaliar a Raspberry Pi 5 como uma unidade de processamento independente, realizando o processamento e a classificação localmente.
- Comparar o desempenho das três arquiteturas em termos de tempo de resposta, latência e eficiência no processamento de dados, utilizando o reconhecimento de gestos como estudo de caso.
- Avaliar as vantagens e desvantagens de cada abordagem, considerando aspectos como desempenho e escalabilidade, com o objetivo de determinar a arquitetura mais adequada para sistemas embarcados em tempo real.

1.2 Organização do texto

Este trabalho está dividido em cinco capítulos, cada um abordando uma etapa específica do desenvolvimento. No Capítulo 1, são apresentados os aspectos gerais sobre a área de estudo, bem como os objetivos da pesquisa. O Capítulo 2 trata dos fundamentos teóricos, abordando os conceitos necessários para o entendimento do tema, incluindo

arquiteturas de sistemas embarcados, FPGAs, SPI, técnicas de processamento de imagens e reconhecimento de gestos, bem como os testes utilizados para avaliação das métricas. No Capítulo 3, é detalhada a metodologia, com a descrição dos equipamentos, ferramentas e procedimentos utilizados na implementação e avaliação das arquiteturas propostas. O Capítulo 4 apresenta os resultados e suas discussões, comparando o desempenho das abordagens e analisando suas vantagens e limitações. Por fim, o Capítulo 5 traz as conclusões do estudo e sugestões para trabalhos futuros, destacando as contribuições do projeto para a área de sistemas embarcados.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os fundamentos teóricos necessários para a compreensão dos principais conceitos abordados. Inicialmente, aborda-se os mais variados sistemas embarcados, com destaque para os microcontroladores, FPGAs e SoCs. Também são discutidos os protocolos de comunicação que viabilizam a integração entre esses dispositivos. Adicionalmente, aborda-se o processamento digital de imagens (PDI) e o reconhecimento de gestos, ilustrando cenários práticos para essas tecnologias. Por fim, abordam-se os testes estatísticos utilizados neste trabalho como alicerce para discussão e avaliação dos resultados.

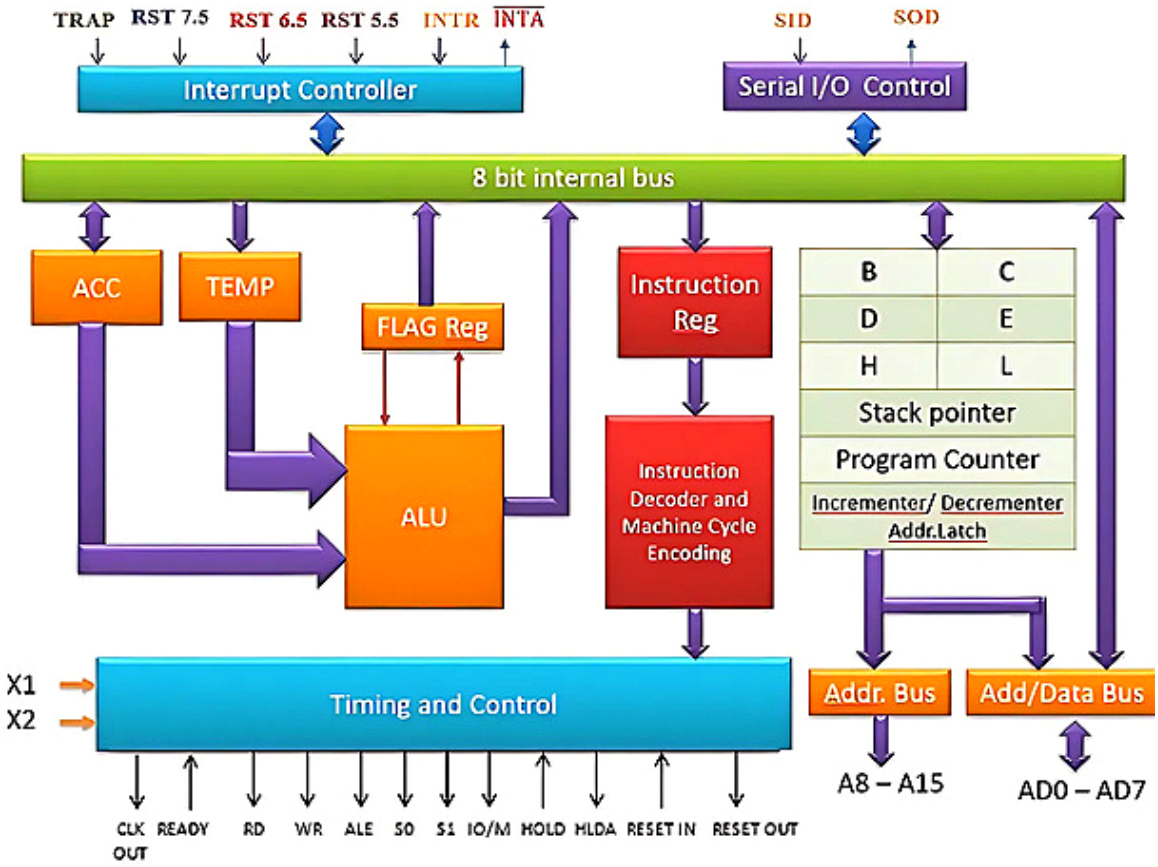
2.1 Fundamentos dos sistemas embarcados: Microprocessadores e Microcontroladores

Sistemas embarcados são dispositivos de computação projetados para desempenhar funções específicas como parte integrante de um sistema maior. Esses sistemas integram componentes essenciais, como hardware, software e, frequentemente, sistemas operacionais em tempo real (*RTOS*), que desempenham um papel fundamental na coordenação e execução precisa e confiável das tarefas, conforme destacado por Kamal (2008).

2.1.1 Microprocessadores

Os microprocessadores são definidos como unidades baseadas em um único chip de integração em larga escala (VLSI), incorporando uma CPU como elemento central. De acordo com Kamal (2008), sua funcionalidade é ampliada por componentes adicionais, como caches e unidades de ponto flutuante, que aceleram o processamento e suportam computações mais complexas. A ilustração presente na figura 2.1 auxilia a visualização da organização interna de um microprocessador, evidenciando a complexidade e a integração de diversos componentes em um único chip.

Figura 2.1: Diagrama geral do microprocessador 8085

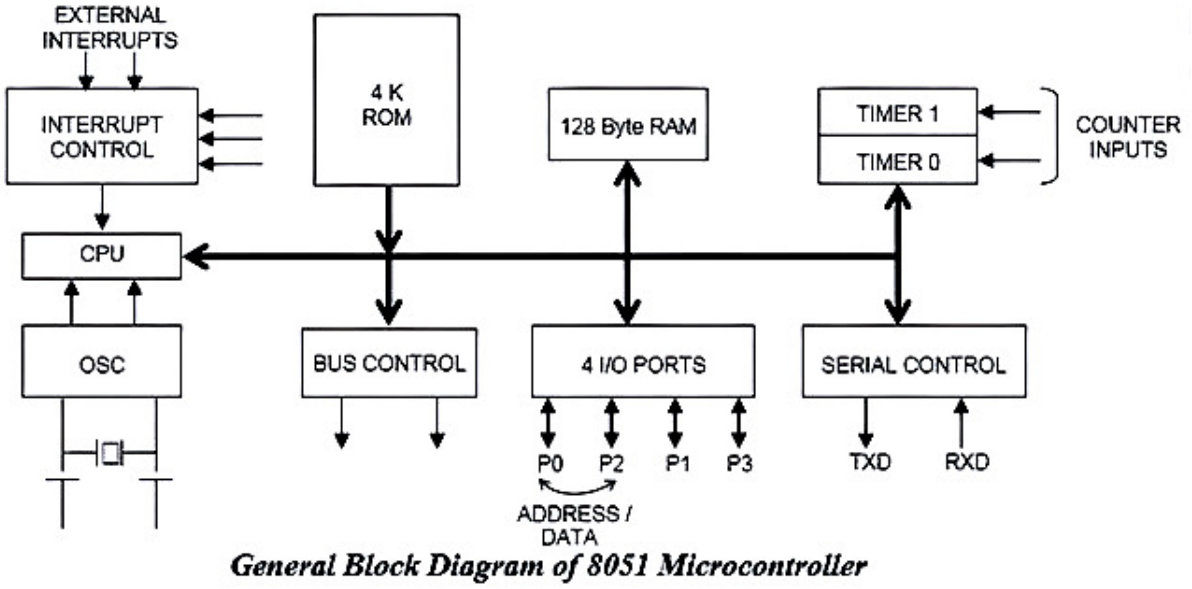


Fonte: M.E (2020)

2.1.2 Microcontroladores

Os microcontroladores são dispositivos integrados projetados para executar funções específicas dentro de um sistema maior. Eles combinam, em um único chip, uma unidade central de processamento (CPU), memória e periféricos de entrada/saída necessários para o controle de sistemas. A Figura 2.2 ilustra essa combinação de componentes em um microcontrolador. Segundo Kamal (2008), esses dispositivos são frequentemente utilizados em sistemas embarcados, especialmente em aplicações de controle e comunicação, operando de forma autônoma. São comumente empregados em sistemas em tempo real, como automação industrial, controle de dispositivos e sistemas de monitoramento.

Figura 2.2: Diagrama geral do microcontrolador 8051



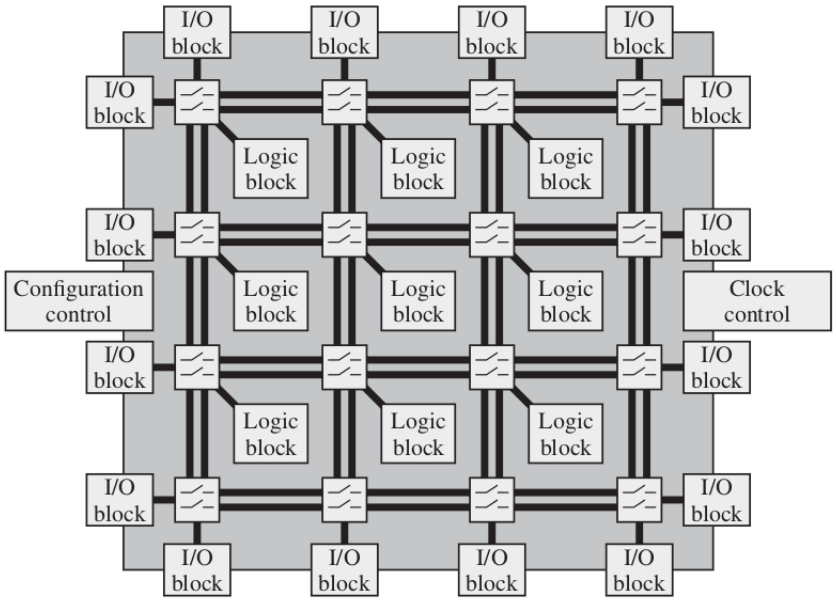
Fonte: Embedded (2017)

Desde a introdução dos microprocessadores nos anos 1970, seguidos pelos microcontroladores na década de 1980, o avanço dessas tecnologias revolucionou o desenvolvimento de sistemas embarcados. Enquanto os microprocessadores continuam sendo a base de computadores de alto desempenho, os microcontroladores impulsionaram a criação de dispositivos de controle dedicados, integrando funcionalidade avançada em chips compactos. Essas tecnologias continuam a evoluir, integrando arquiteturas mais eficientes e recursos que atendem às crescentes demandas por sistemas embarcados inteligentes e conectados.

2.2 FPGAs *(Field Programmable Gate Arrays)*

Field Programmable Gate Arrays (FPGAs) são dispositivos semicondutores configuráveis, compostos por blocos lógicos interconectados que podem ser programados para realizar tarefas específicas, ver Figura 2.3. Ao contrário de circuitos integrados fixos, as FPGAs permitem flexibilidade no desenvolvimento de hardware, adaptando-se a diferentes aplicações sem a necessidade de redesenhar fisicamente o chip. O funcionamento de uma FPGA é viabilizado por meio de linguagens de descrição de hardware (*HDLs*), como *VHDL* ou *Verilog*. Essas linguagens são usadas para modelar circuitos digitais, permitindo a criação de implementações adequadas para requisitos específicos.

Figura 2.3: Elementos de um FPGA



Fonte: Adaptado de Bailey (2023)

O funcionamento base do hardware reprogramável é ter um circuito genérico, cuja funcionalidade pode ser configurada para uma aplicação específica. Como Bailey (2023) descreve, os sistemas baseados em ALU (Unidades Lógicas Aritméticas) realizam uma operação por vez, limitando sua capacidade para tarefas complexas. Em contraste, a lógica programável implementa funcionalidades de forma paralela, o que oferece maior desempenho, especialmente para tarefas exigentes, como processamento digital de sinais e imagens. As FPGAs, por exemplo, permitem essa flexibilidade, oferecendo baixa latência e alta capacidade de personalização para aplicações específicas. Além disso, eles apresentam baixa latência em comparação com processadores tradicionais, como CPUs e GPUs, em especial para tarefas de processamento de imagem, como observado em Asano et al. (2009).

No entanto, há desvantagens associadas ao uso de FPGAs. O custo inicial dos dispositivos e ferramentas de desenvolvimento pode ser elevado. Além disso, o processo de programação exige conhecimento técnico especializado, o que pode limitar sua adoção por profissionais menos experientes.

2.3 SoCs (System on Chip)

Os *Systems-on-Chip* (*SoCs*) são dispositivos que integram em um único chip múltiplos componentes essenciais de um sistema eletrônico, como processadores, memória, interfaces de comunicação e periféricos, conforme indicado em Kamal (2008). Essa abordagem permite desenvolver sistemas compactos, eficientes em consumo de energia e com

alto desempenho, tornando os SoCs uma solução ideal para dispositivos embarcados e aplicações de alta integração. Uma característica notável dos SoCs modernos é sua capacidade de suportar arquiteturas híbridas, combinando a flexibilidade de FPGAs com a eficiência de hardware especializado, além de interconexões otimizadas, como o barramento AXI.

2.3.1 Arquiteturas híbridas em sistemas embarcados

As arquiteturas híbridas em sistemas embarcados combinam diferentes paradigmas de processamento em uma única plataforma, como processamento sequencial, paralelo, reconfigurável e acelerado. Essas arquiteturas integram tecnologias como CPUs de propósito geral, processadores digitais de sinal (*DSPs*), aceleradores especializados, e outros elementos que trabalham em conjunto para atender às necessidades específicas de aplicações embarcadas, conforme observado em Kamal (2008). O objetivo principal dessas arquiteturas é equilibrar flexibilidade, eficiência energética e desempenho, adaptando-se tanto a cargas de trabalho variáveis quanto a requisitos críticos, como baixa latência ou alta taxa de transferência.

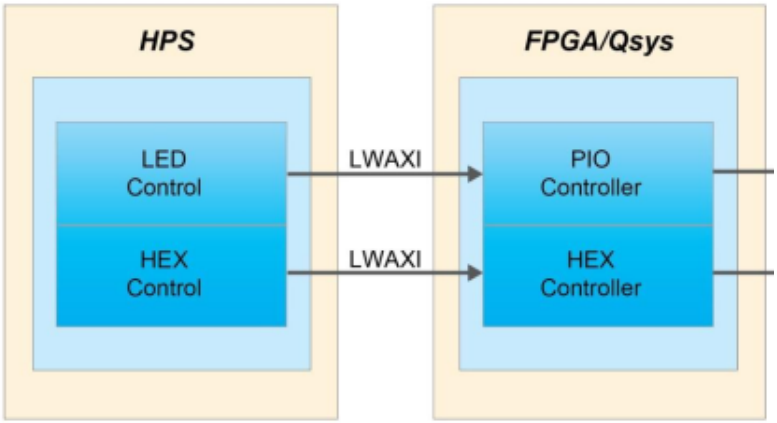
Em arquiteturas híbridas, processadores de propósito geral costumam ser responsáveis por tarefas mais complexas e variáveis, enquanto unidades especializadas, como aceleradores, lidam com funções específicas, incluindo criptografia, codificação de vídeo ou processamento de sinais. Essa divisão de responsabilidades melhora a utilização dos recursos de hardware e reduz o tempo necessário para a execução de operações críticas, tornando o sistema mais eficiente e adequado para lidar com múltiplas demandas simultâneas.

2.3.2 Arquiteturas híbridas - FPGAs

As *FPGAs* desempenham um papel fundamental no design de arquiteturas híbridas, oferecendo flexibilidade e alto desempenho para sistemas embarcados. Essas arquiteturas combinam a capacidade de reconfiguração das FPGAs com processadores tradicionais, como CPUs, criando plataformas capazes de executar tanto tarefas sequenciais quanto operações paralelas e aceleradas.

A arquitetura híbrida presente no DE1-SoC, ilustrada na Figura 2.4, exemplifica essa integração. Nesse dispositivo, um FPGA é combinado com um HPS, neste caso o ARM Cortex-A9, em um único SoC (System-on-Chip). Essa configuração permite que o processador ARM gerencie o sistema operacional e tarefas gerais, enquanto o FPGA é utilizado para executar operações altamente paralelas ou personalizadas, como algoritmos de processamento de sinais, criptografia ou aceleração de aprendizado de máquina.

Figura 2.4: Arquitetura híbrida presente no Cyclone V SoC do DE1-SoC



Fonte: Adaptado de Terasic Inc. (2019a)

2.3.3 Interfaces AXI e Bridges em arquiteturas híbridas

Em sistemas embarcados que combinam *FPGAs* e *Hard Processor Systems* (*HPS*), a comunicação eficiente entre essas unidades é essencial para maximizar o desempenho e a flexibilidade. Interfaces baseadas no protocolo AXI (*Advanced eXtensible Interface*) e *bridges* especializados são amplamente utilizadas para facilitar essa interação, conforme indicado em Intel Corporation (2013), permitindo que o *FPGA* e o *HPS* troquem dados de forma otimizada.

Interface AXI

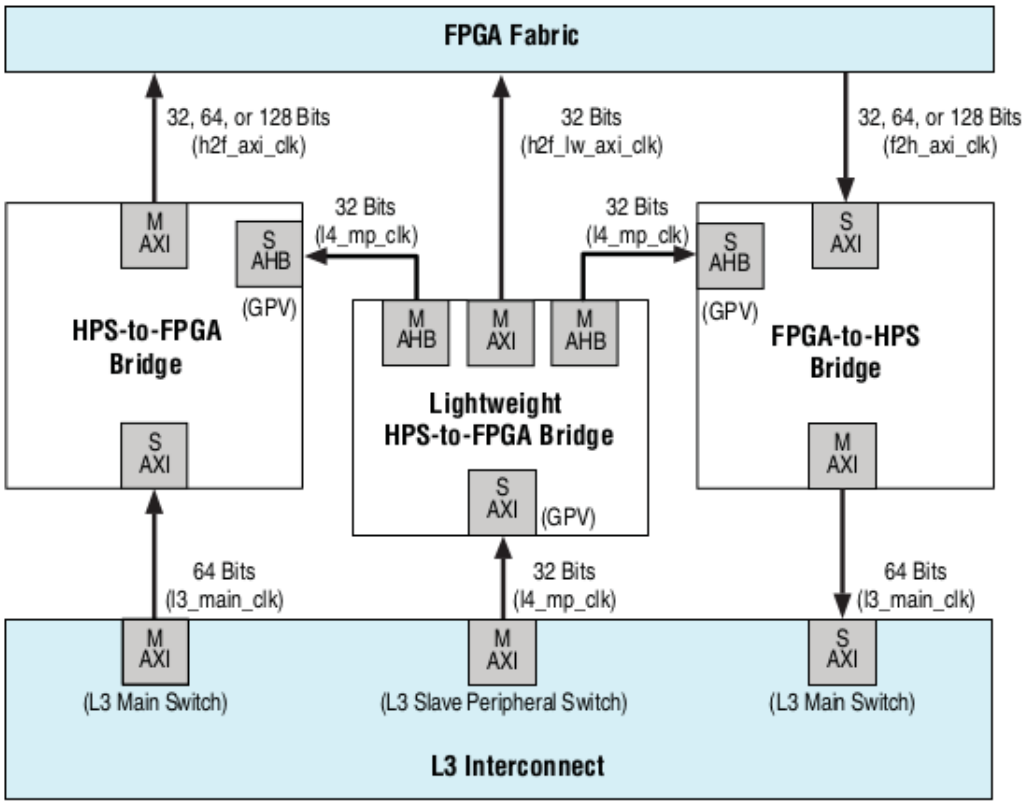
A interface *AXI* (*Advanced eXtensible Interface*), integrante do padrão *AMBA* (*Advanced Microcontroller Bus Architecture*), oferece suporte para comunicações de alta largura de banda, baixa latência e transações paralelas. Essa interface é fundamental para conectar diferentes componentes de sistemas heterogêneos, como o *FPGA* e o *HPS*, permitindo que tarefas de controle, processamento e troca de grandes volumes de dados sejam realizadas de maneira eficiente e escalável.

Comunicação *HPS-FPGA* por meio da interface *AXI*

Conforme descrito em Intel Corporation (2013), as *interfaces* (ou bridges) *AXI* entre o *HPS* e o *FPGA* permitem a comunicação bidirecional entre as lógicas do *FPGA* e do *HPS*. Esses bridges consistem em pares de *interfaces AXI master-slave*, com uma interface exposta à lógica do *FPGA* e a outra à lógica do *HPS*. O *FPGA-to-HPS Bridge* expõe uma interface *AXI slave*, permitindo que componentes no *FPGA* acessem a memória ou periféricos no *HPS*. Já os *HPS-to-FPGA* e *Lightweight HPS-to-FPGA Bridges* expõem *interfaces master AXI*, possibilitando que a lógica do *HPS* acesse memórias ou periféricos

no *FPGA*, ver Figura 2.5

Figura 2.5: Barramento e conexão das interfaces *AXI* em um sitema *HPS-FPGA*



Fonte: Adaptado de Intel Corporation (2013)

2.4 Protocolos de comunicação em sistemas embarcados

Os sistemas embarcados frequentemente necessitam trocar dados com outros dispositivos, sensores, ou componentes internos. Para isso, são utilizados interfaces ou protocolos de comunicação, que definem as regras e padrões para a transmissão de informações entre os sistemas, conforme indicado por Bailey (2023). Esses protocolos podem ser classificados em protocolos seriais e protocolos paralelos, com os protocolos seriais sendo mais comuns em sistemas embarcados devido à sua simplicidade e menor consumo de pinos de comunicação. Além disso, esses protocolos variam em termos de velocidade de transmissão, confiabilidade e complexidade, sendo escolhidos conforme os requisitos do sistema em questão.

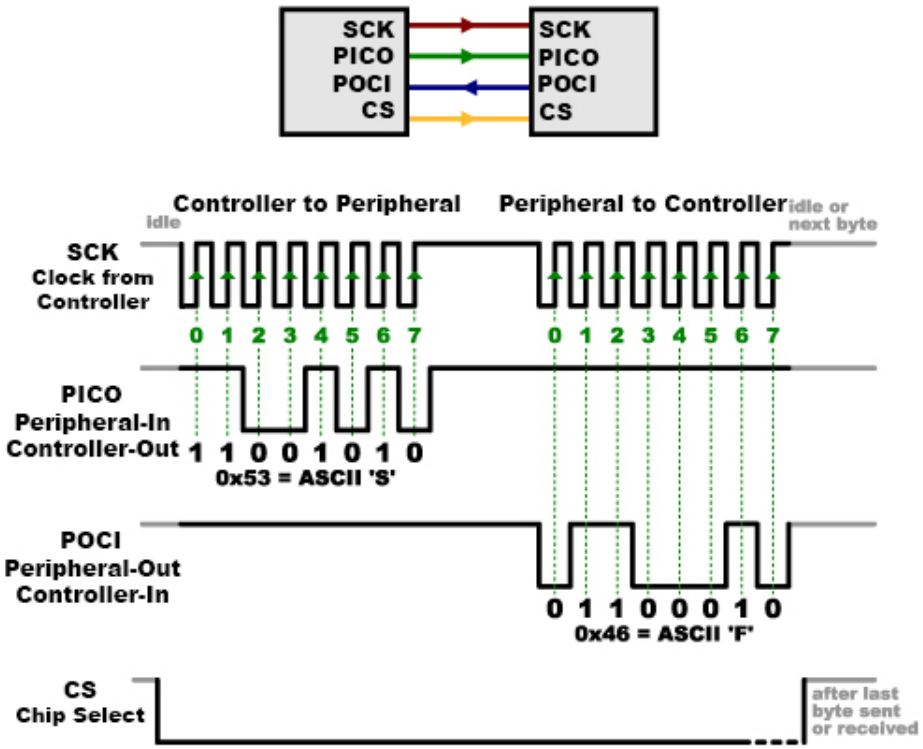
2.4.1 *SPI (Serial Peripheral Interface)*

O *SPI* (Serial Peripheral Interface) é um dos protocolos seriais mais utilizados em sistemas embarcados. Ele é projetado para comunicação de alta velocidade entre microcontroladores e periféricos, como sensores, memórias *flash* e *displays*, com a vantagem de ser simples e eficiente, conforme observado em Staszewski et al. (2018). O *SPI* utiliza um modelo mestre-escravo, muitas vezes referenciado como controlador-periférico, onde um dispositivo mestre controla a comunicação e um ou mais dispositivos escravos respondem às solicitações do mestre. O *SPI* funciona com quatro sinais principais, sendo eles:

- ***MOSI (Master Out Slave In) ou PICO (Peripheral In Controller Out)***: Linha de dados do mestre ou controlador para o escravo ou periférico.
- ***MISO (Master In Slave Out) ou POCI (Peripheral Out Controller In)***: Linha de dados do escravo, ou periférico, para o mestre, ou controlador.
- ***SCLK (Serial Clock) ou SCK (Serial Clock)***: Linha de *clock* gerada pelo mestre ou controlador, que sincroniza a comunicação.
- ***SS/CS (Slave Select/Chip Select)***: Linha que seleciona o escravo, ou periférico, ativo. O mestre escolhe qual escravo irá se comunicar, ativando o sinal correspondente.

A comunicação *SPI* é *full-duplex*, ou seja, dados podem ser transmitidos e recebidos simultaneamente, o que a torna mais eficiente em comparação com outros protocolos como o I2C, o qual é *half-duplex*. A velocidade de transmissão no *SPI* pode ser bastante alta, dependendo das capacidades do hardware envolvido. A Figura 2.6 ilustra o funcionamento básico de uma comunicação SPI, destacando os sinais principais e o fluxo de dados entre o mestre e o escravo.

Figura 2.6: Funcionamento geral do protocolo SPI



Fonte: SparkFun (2024)

2.4.2 Limitações do *SPI*

Embora o *SPI* seja amplamente utilizado, ele apresenta algumas limitações que devem ser consideradas ao ser implementado em sistemas embarcados. Uma das principais restrições está no número de dispositivos que podem ser conectados. Para cada escravo adicional, é necessário um pino de seleção dedicado, o que pode resultar em um grande consumo de pinos em sistemas com muitos periféricos, limitando sua escalabilidade. Além disso, o *SPI* é eficaz apenas em curtas distâncias, ressalta PiEmbSysTech (2024). A qualidade do sinal de *clock* se perde em distâncias maiores, o que pode comprometer a comunicação, tornando-o menos adequado para sistemas que exigem maior alcance. O protocolo também consome mais pinos em comparação com outros protocolos seriais, como o *I2C*, o que pode ser um problema em sistemas com recursos limitados de entradas e saídas.

No que diz respeito à operação, o *SPI* não possui um sistema nativo de endereçamento, o que torna a comunicação com múltiplos dispositivos mais complexa, exigindo que o mestre gerencie manualmente a seleção dos escravos. Além disso, embora seja possível conectar vários dispositivos escravos, o processo de gerenciamento da comunicação entre eles aumenta a complexidade do sistema. Por fim, embora o SPI ofereça

altas taxas de transmissão, a velocidade pode ser afetada por fatores como a qualidade do sinal e a distância percorrida pelo mesmo entre os dispositivos, limitando sua eficácia em ambientes que exigem uma comunicação muito rápida ou em longas distâncias.

2.5 Processamento digital de imagens (PDI)

O processamento digital de imagens (PDI) refere-se à manipulação de imagens digitais para extrair informações úteis ou transformá-las de acordo com um objetivo específico. Segundo Gonzalez (2017), essas manipulações envolvem operações como melhoria de contraste, remoção de ruídos, segmentação para identificação de regiões ou objetos, e extração de atributos relevantes, como bordas e texturas. O PDI é estruturado em diferentes níveis de processamento, desde tarefas básicas até etapas mais complexas, como classificação de padrões. Nos tópicos subsequentes, serão exploradas técnicas como segmentação, limiarização, métodos para realçar características visuais e operações morfológicas.

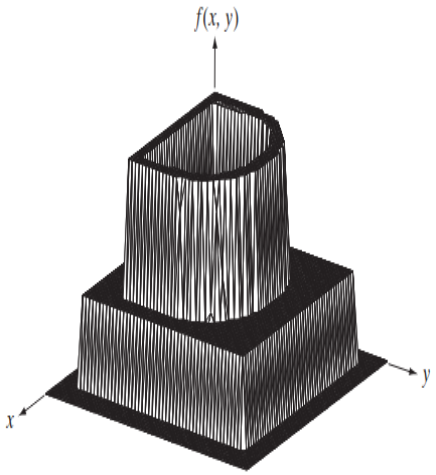
2.5.1 Imagens digitais

Segundo Bailey (2023), uma imagem digital é uma representação discreta de uma cena ou objeto, organizada como uma matriz bidimensional (ou de maior dimensão) de valores numéricos representados por *bytes*. Esses valores, conhecidos como pixeis em imagens 2D ou *voxels* em imagens 3D, resultam de um processo de amostragem espacial, onde uma função contínua é convertida em quantidades discretas. A quantização atribui valores inteiros a cada ponto de amostra, permitindo a representação de diferentes intensidades e características da imagem, como cores ou texturas.

Esse processo de amostragem resulta, em muitos casos, em uma matriz 2D, $f(x, y)$, com M linhas e N colunas, onde x e y são as coordenadas discretas, conforme descrito por Gonzalez (2017). Além disso, essas coordenadas são expressas como valores inteiros, com $0 \leq x \leq M-1$ e $0 \leq y \leq N-1$. O valor de cada ponto da imagem digital na coordenada (x, y) é representado por $f(x, y)$, onde x e y são inteiros, conforme a Equação 2.1. Esse conjunto de coordenadas forma o domínio espacial da imagem e os valores resultantes de amostragem são tratados como variáveis espaciais, permitindo o processamento digital da imagem, como ilustrado na figura 2.7.

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

Figura 2.7: Exemplos de representação de uma imagem digital



(a) Exemplo de imagem exibida como superfície

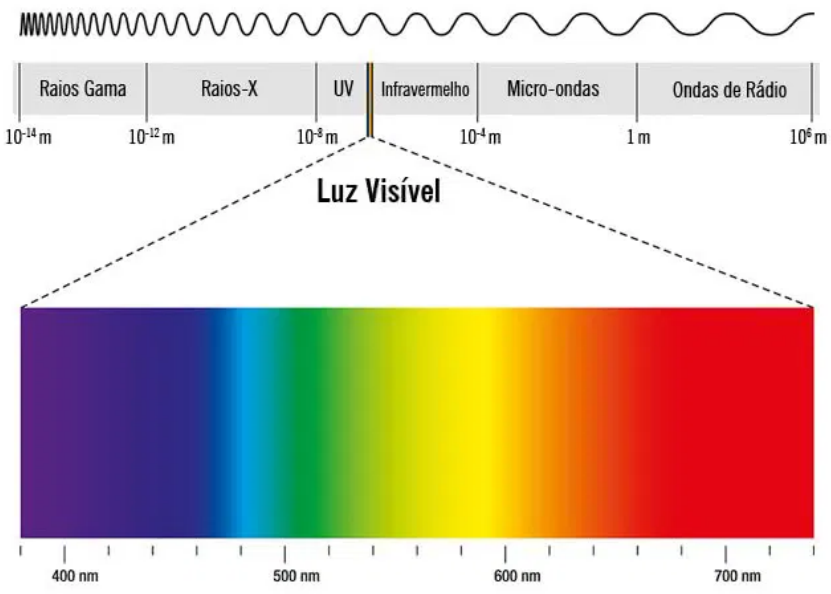
(b) Exemplo de imagem como matriz de intensidades

Fonte: Adaptado de Gonzalez (2017)

2.5.2 Espaços de cores

A cor é uma percepção visual resultante da interação da luz com objetos e sua interpretação pelo sistema visual humano. Essa percepção é influenciada pelas características da luz refletida ou emitida pelos objetos, como comprimento de onda, intensidade e saturação, ver Figura 2.8.

Figura 2.8: Espectro eletromagnético visível ao olho humano



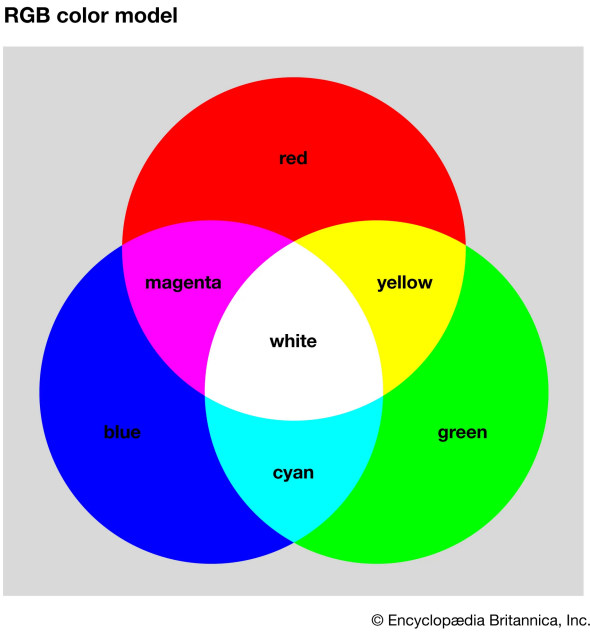
Fonte: Toda Matéria (2024)

Em imagens digitais, a cor é representada numericamente em espaços de cores, que organizam informações em componentes numéricas. Os espaços de cores são sistemas utilizados para organizar e descrever cores numericamente em imagens digitais. Eles permitem representar cores consistentemente em diferentes dispositivos e aplicações. Cada espaço de cor é projetado para atender a necessidades específicas, como precisão visual, compressão de dados ou eficiência computacional.

Espaço de cor RGB

O espaço de cores RGB (Red, Green, Blue) é um modelo aditivo de cores baseado na mistura de três componentes primárias de luz: vermelho, verde e azul. Cada cor é representada por intensidades dessas componentes, variando de 0 a 255 em representações digitais comuns, permitindo a reprodução de uma ampla gama de cores, ver Figura 2.9. Esse espaço é amplamente utilizado em dispositivos eletrônicos, como monitores e câmeras, devido à sua correspondência com a percepção humana da luz e facilidade de manipulação em sistemas computacionais.

Figura 2.9: Espaço de cor RGB e suas mesclas



Fonte: Zelazko (2024)

Espaço de cor YCbCr

O espaço de cor YCbCr é utilizado para representar imagens separando a informação de brilho (Y) das informações de cor (Cb e Cr). A componente Y reflete a luminância, ou seja, o brilho da imagem, enquanto Cb e Cr representam as diferenças de cor em relação ao brilho, com Cb associado à cor azul e Cr à cor vermelha, ver Fi-

gura 2.10. Essa separação facilita a compressão de imagens e vídeos, aproveitando que o olho humano é mais sensível à luminância do que à crominância, permitindo uma maior eficiência no armazenamento e transmissão de dados.

Figura 2.10: Espaço de cor YCbCr. À esquerda: Canal de luminância. Ao centro: Canal de Crominância Azul. À direita: Canal de Crominância Vermelha

Fonte: Adaptado de Bailey (2023)

2.5.3 Compensação de iluminação

A compensação de iluminação em imagens digitais é uma técnica utilizada para ajustar os níveis de brilho e contraste de uma imagem, corrigindo variações de iluminação que podem ocorrer durante a captura da imagem. Essas variações podem resultar em áreas muito escuras (subexpostas) ou muito claras (superexpostas), prejudicando a visibilidade e a análise da imagem. A compensação de iluminação pode ser realizada utilizando técnicas como equalização de histograma, que redistribui os níveis de intensidade de forma mais uniforme, ou ajustes de contraste, que melhoram a percepção visual das imagens. Para aumentar o brilho de uma imagem, é comum adicionar uma constante ao valor dos píxeis, enquanto para escurecer, subtrai-se uma constante. Segundo Bailey (2023), a equação 2.2 representa a função de mapeamento responsável por realizar operações ponto a ponto na imagem, ou seja, aplica uma transformação f a cada píxel da imagem de entrada $I[x, y]$ para gerar os valores correspondentes na imagem de saída $Q[x, y]$.

$$Q[x, y] = f(I[x, y]) \tag{2.2}$$

$$Q = a(I + b') \tag{2.3}$$

Na equação 2.3, a transformação é controlada por parâmetros que afetam propriedades da imagem, como contraste e brilho. O contraste da imagem, por exemplo, é afetado pela inclinação da função de mapeamento. Os parâmetros da equação, bem como seus efeitos podem ser descritos abaixo:

- a - controla a inclinação da função, ajustando diretamente o contraste da imagem,

na qual $a > 1$ amplifica as diferenças entre os valores de intensidade dos píxeis, aumentando o contraste. Enquanto $a < 1$ reduz essas diferenças, diminuindo o contraste.

- b' - é um deslocamento que altera o brilho inicial da imagem antes da aplicação do ajuste de contraste.

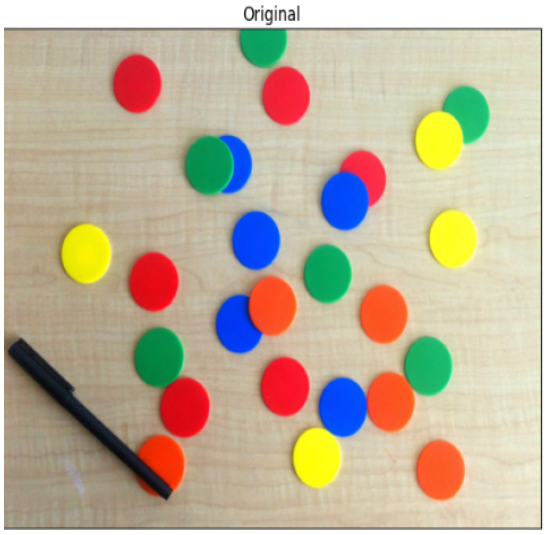
2.5.4 Segmentação de imagens

A segmentação de imagens é uma técnica fundamental no processamento digital de imagens, que consiste em dividir uma imagem em regiões com base em características específicas, como intensidade, textura, cor ou outros atributos visuais, com o objetivo de isolar áreas relevantes para análise e interpretação, conforme indicado por Gonzalez (2017).

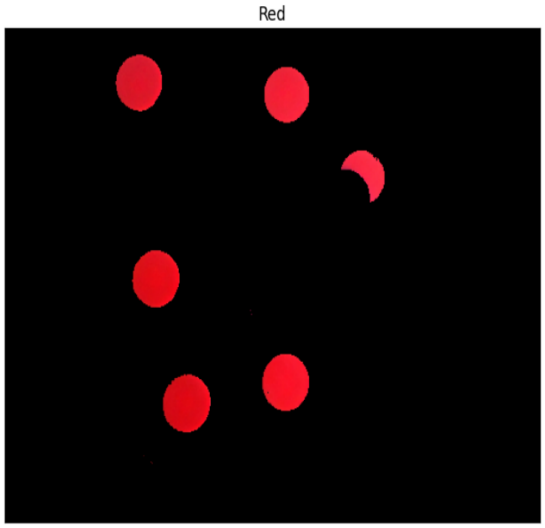
Segmentação por cor

A técnica que utiliza informações cromáticas para identificar e separar regiões de uma imagem é amplamente conhecida por segmentação por cor. Essa abordagem baseia-se na análise das componentes de cor de cada píxel, frequentemente representadas em espaços de cor como RGB (vermelho, verde, azul) ou YCbCr (luminância, crominância azul e vermelha). Ao definir intervalos específicos para as características de cor, é possível identificar regiões que compartilham propriedades cromáticas similares, permitindo o isolamento de objetos ou áreas de interesse, conforme indicado por Gonzalez (2017). Essa técnica é amplamente aplicada em cenários onde as diferenças de cor entre as regiões são suficientes para realizar uma segmentação precisa, como ilustrado na Figura 2.11.

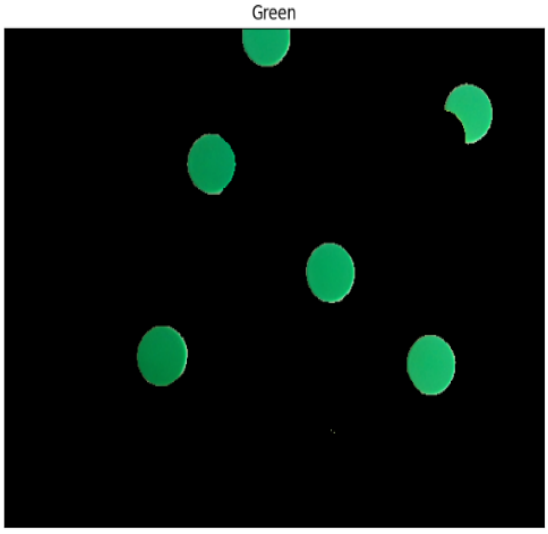
Figura 2.11: Exemplos de segmentação por cor em uma imagem digital colorida



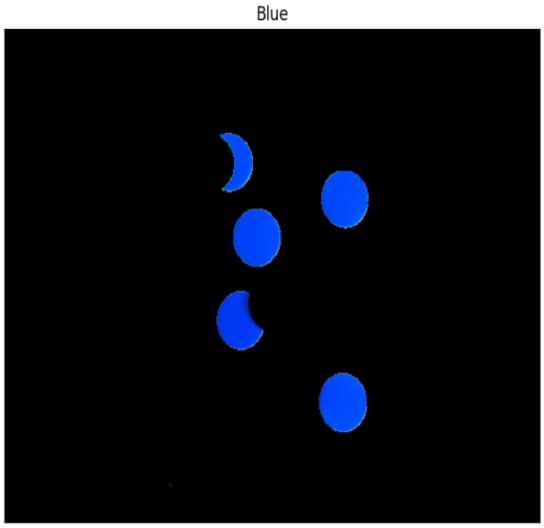
(a) Exemplo de imagem colorida



(b) Segmentação de tons de vermelho



(c) Segmentação de tons de verde



(d) Segmentação de tons de azul

Fonte: Autoria própria

Limiarização e binarização

A limiarização é uma técnica de segmentação baseada em valores de intensidade de píxeis. Segundo Gonzalez (2017) a operação consiste em aplicar um valor limiar (*threshold*) para classificar os píxeis de uma imagem em diferentes categorias, como objeto e fundo. A equação 2.4 descreve a operação supracitada na imagem, onde $0 \leq T \leq 255$.

$$g(x,y) = \begin{cases} 255 & \text{, se } f(x,y) > T \\ 0 & \text{, se } f(x,y) \leq T \end{cases} \tag{2.4}$$

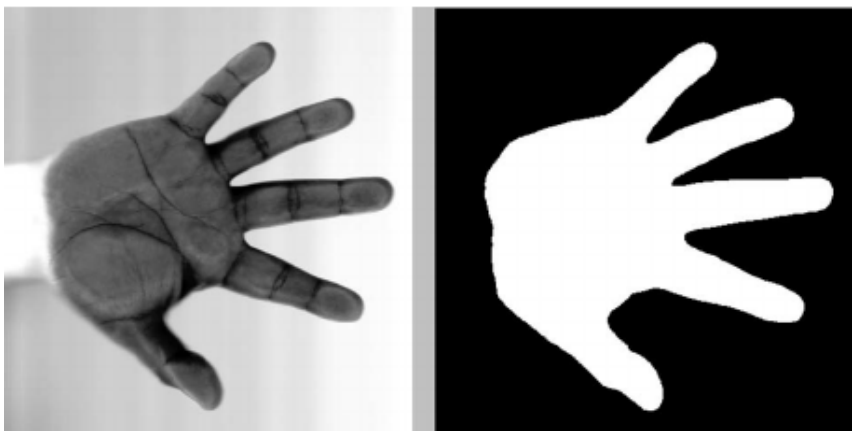
Um caso específico de limiarização é a **binarização**, onde o valor limiar divide

os píxeis em duas classes: valores acima do limiar são atribuídos a uma classe (geralmente representada pelo branco) e valores abaixo, à outra (representada pelo preto). Essa operação é amplamente utilizada em pré-processamento de imagens para destacar objetos de interesse em relação ao fundo, especialmente em imagens em tons de cinza. A Figura 2.12 ilustra a aplicação de uma limiarização em várias classes, enquanto a Figura 2.13 apresenta um exemplo de binarização.

Figura 2.12: Exemplo do processo de limiarização. À esquerda: Imagem original com ruídos. À direita: Limiarização multinível em uma imagem em tons de cinza.

Fonte: Adaptado de Arêdes (2009)

Figura 2.13: Exemplo de binarização como um caso específico de limiarização. À esquerda: Imagem original. À direita: Imagem binarizada



Fonte: Gomez-Gil (2009)

2.5.5 Operações morfológicas

As operações morfológicas são técnicas utilizadas no processamento de imagens digitais para análise e manipulação de formas e estruturas presentes em imagens, especialmente em imagens binárias. Baseadas na teoria de conjuntos, essas operações utilizam

um elemento estruturante, o qual é uma matriz de valores, para interagir com a imagem e transformar suas características. Os dois operadores morfológicos fundamentais são a erosão e a dilatação, que servem como base para operações mais complexas, como abertura, fechamento, gradiente morfológico, entre outras.

Segundo Gonzalez (2017), as operações morfológicas são amplamente aplicadas em tarefas como segmentação de objetos, remoção de ruído, análise de bordas e preenchimento de lacunas em imagens binárias. Essas técnicas modificam a imagem com base na interação entre suas estruturas e o elemento estruturante, permitindo a análise ou transformação de características conforme sua forma e tamanho.

Erosão

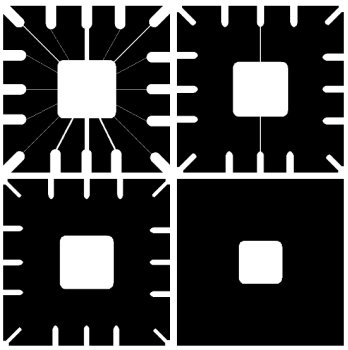
A erosão é uma operação morfológica que reduz ou elimina regiões brancas (objetos) em uma imagem binária, dependendo do elemento estruturante. A operação consiste em deslizar o elemento estruturante sobre a imagem e apagar os píxeis da região-alvo onde ele não se ajusta completamente. Em termos práticos, a erosão encolhe as formas e pode desconectar componentes pequenos ou estreitar bordas.

Conforme descrito por Gonzalez (2017), a erosão é definida matematicamente como:

$$A \ominus B = \{z \mid B_z \subseteq A\}$$

onde A é o conjunto de píxeis da imagem, B é o elemento estruturante, e B_z representa a translação de B para a posição z. A operação mantém os píxeis apenas onde B está completamente contido em A, conforme Figura 2.14.

Figura 2.14: Operação de erosão. (Cima esquerda) Imagem binária de uma *wire-bond mask*. (Cima direita) Imagem erodida usando um elemento estruturante quadrado de tamanho 3. (Baixo esquerda) Imagem erodida usando um elemento estruturante quadrado de tamanho 5. (Baixo direita) Imagem erodida usando um elemento estruturante quadrado de tamanho 7.



Fonte: Gonzalez (2017)

Dilatação

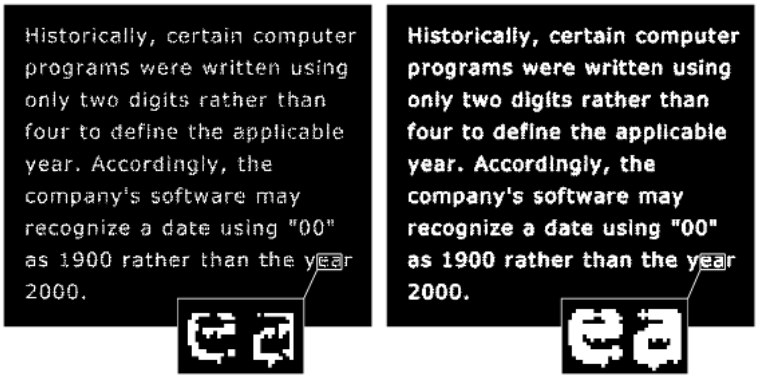
A dilatação, por sua vez, é uma operação morfológica que, ao contrário da erosão, expande as regiões brancas (objetos) em uma imagem binária, também com base no elemento estruturante. Essa expansão ocorre ao adicionar píxeis às bordas das regiões-alvo, conforme o elemento estruturante interage com a imagem como mostra a Figura 2.15. A dilatação pode conectar componentes próximos e preencher lacunas em estruturas.

De acordo com Gonzalez (2017), a dilatação é matematicamente expressa como:

$$A \oplus B = \{z \,|\, (\hat{B})_z \cap A) \neq \emptyset\}$$

Nesse caso, A é o conjunto de píxeis da imagem e $(\hat{B})_z$ é o elemento estruturante refletido em sua própria origem. A operação adiciona píxeis onde B sobrepõe A, permitindo o crescimento das formas na imagem.

Figura 2.15: Operação de dilatação. À esquerda: Texto de exemplo com caracteres ruidosos. À direita: Imagem após dilatação



Fonte: Gonzalez (2017)

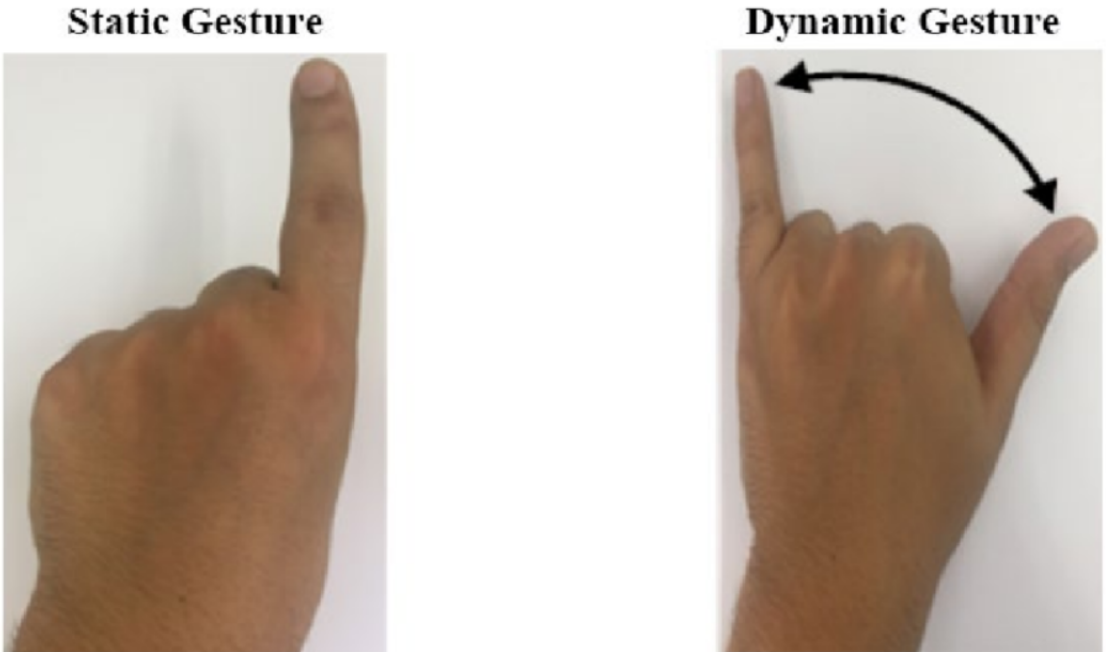
2.6 Reconhecimento de gestos

O reconhecimento de gestos é a interpretação de movimentos manuais ou corporais para fornecer comandos a sistemas computacionais. Essa tecnologia permite a interação direta e natural com dispositivos, substituindo e/ou complementando o uso de periféricos tradicionais (câmeras, teclados, controles). Através da detecção de gestos específicos, como movimentos das mãos ou posturas corporais, sistemas computacionais podem realizar tarefas, melhorar a navegação e facilitar o controle em ambientes virtuais, tornando a interação mais intuitiva e acessível.

2.6.1 Classificações de gestos

Como afirmado por Jeon et al. (2024), a maioria dos estudos existentes considera como principais duas categorias de gestos: gestos com movimento (dinâmicos) e gestos sem movimento (estáticos), ver Figura 2.16. Gestos com movimento são caracterizados pela alteração da posição do corpo ou de uma parte do corpo, como o movimento das mãos, braços ou cabeça. Já os gestos sem movimento referem-se a ações que não envolvem deslocamento físico, como a ativação de comandos por meio de posturas específicas ou mudanças na orientação do corpo (gestos estáticos).

Figura 2.16: Exemplos de gestos estático e dinâmico



Fonte: Adaptado de Al-Shamayleh et al. (2020)

A relevância do reconhecimento de gestos está associada à crescente demanda por interfaces homem-máquina mais acessíveis e eficientes. Em aplicações como acessibilidade, sistemas de automação residencial e dispositivos vestíveis, ele se destaca como uma ferramenta importante para ampliar a usabilidade e inclusão tecnológica.

2.6.2 Aplicações e dificuldades

O reconhecimento de gestos tem aplicações amplas em diferentes contextos tecnológicos. Em dispositivos *IoT* (*Internet of Things*), por exemplo, ele pode ser utilizado para controlar eletrodomésticos ou sistemas de iluminação por meio de gestos simples, como evidenciado em Torres et al. (2024). Nas áreas de realidade aumentada e virtual (AR/VR), por exemplo, gestos são usados para navegação em ambientes imersivos, facilitando interações que simulam a manipulação direta de objetos. Contudo, as limitações

computacionais, como apontado por Luo (2022), especialmente em dispositivos embarcados, representam um desafio para a execução de algoritmos em tempo real. Soluções que demandam elevado poder de processamento e memória, como redes neurais profundas, muitas vezes requerem otimizações para rodar eficientemente em plataformas de hardware de baixo custo, favorecendo o uso de técnicas tradicionais de processamento de imagens.

2.7 Testes estatísticos para avaliação das métricas

Os testes estatísticos são ferramentas fundamentais para validar hipóteses em dados experimentais, permitindo determinar se uma observação é ou não estatisticamente significativa. Através da comparação de uma hipótese nula (que afirma, de certo modo, não haver efeito ou diferença) com dados observados, esses testes ajudam a concluir se a diferença observada é atribuída ao acaso ou se reflete uma real diferença entre os grupos analisados. Em experimentos científicos, os testes estatísticos são essenciais para garantir que as conclusões tiradas dos dados são confiáveis e não ocorrem apenas por variação aleatória, como destacado por Montgomery (2001).

2.7.1 Teste *t* de *Student*

O teste t de Student é utilizado para comparar as médias de dois grupos independentes com o objetivo de verificar se há diferenças estatisticamente significativas entre essas médias. Ele avalia a diferença entre as médias dos grupos em relação à variabilidade observada nos dados. A hipótese nula do teste t afirma que as médias dos dois grupos são iguais, enquanto a hipótese alternativa sugere que as médias são diferentes. Esse teste é apropriado quando os dados seguem uma distribuição normal e as variâncias são homogêneas.

Hipóteses do teste t:

- Hipótese nula (H0): As médias dos dois grupos são iguais.
- Hipótese alternativa (H1): As médias dos dois grupos são diferentes.

A estatística de teste t é calculada pela fórmula:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Onde:

- $\bar{X}_1$ e $\bar{X}_2$ são as médias dos dois grupos.
- s_1^2 e s_2^2 são as variâncias dos dois grupos.

- n_1 e n_2 são os tamanhos das amostras dos grupos 1 e 2, respectivamente.

O valor-p é obtido consultando a distribuição t com os graus de liberdade calculados. O valor-p corresponde à probabilidade de observar um valor t tão extremo (ou mais extremo) quanto o obtido, assumindo que a hipótese nula é verdadeira. Se o p-valor obtido do teste for menor que o nível de significância (geralmente 0.05), rejeita-se a hipótese nula, indicando haver uma diferença significativa entre as médias dos dois grupos, como indicado em Harris (2015a).

2.7.2 Teste de Kruskal-Wallis

O teste de Kruskal-Wallis é um teste não-paramétrico e utilizado quando os dados não seguem uma distribuição normal. Ao contrário de testes paramétricos, como o teste t, que comparam médias sob pressupostos de normalidade e homogeneidade de variâncias, o teste de Kruskal-Wallis avalia diferenças entre grupos com base nos ranks dos dados. É apropriado para situações em que os pressupostos de normalidade ou homogeneidade de variâncias são violados. A hipótese nula estabelece que todas as distribuições dos grupos são equivalentes, enquanto a hipótese alternativa indica a existência de pelo menos uma distribuição distinta.

Hipóteses do Kruskal-Wallis:

- Hipótese nula (H0): As distribuições dos grupos são iguais, ou seja, as medianas dos grupos são iguais.
- Hipótese alternativa (H1): Pelo menos uma das distribuições dos grupos é diferente.

A estatística de teste H do Kruskal-Wallis é dada por:

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(N+1)$$

Onde:

- N é o número total de observações.
- k é o número de grupos.
- R_i é a soma dos ranks do grupo i.
- n_i é o número de observações no grupo i.

O valor de H é comparado com a distribuição qui-quadrado (χ^2) com $k-1$ graus de liberdade. Caso o p-valor associado a H seja inferior ao nível de significância, rejeita-se a hipótese nula, indicando que existem diferenças significativas entre as distribuições dos grupos, conforme indicado em Harris (2015b).

2.7.3 Escolha dos testes

A escolha dos testes estatísticos depende das características dos dados. O *teste t* para amostras independentes foi utilizado para comparar as médias de dois grupos, sob a suposição de normalidade e variâncias homogêneas. Quando essas suposições não são atendidas, o teste de Kruskal-Wallis, uma alternativa não paramétrica, foi empregado para comparar as distribuições por meio de ranks.Essa abordagem visa avaliar adequadamente as diferenças entre os grupos conforme as características dos dados.

Capítulo 3

Metodologia

3.1 Infraestrutura e ferramentas utilizadas

Nesta seção são apresentados os recursos empregados no desenvolvimento do experimento, considerando os dispositivos de hardware necessários e as ferramentas de software aplicadas à configuração e implementação.

3.1.1 Equipamentos

O estudo contou com uma infraestrutura composta por diversos dispositivos e equipamentos fundamentais para a execução do estudo. Entre os principais itens utilizados estão: o Kit de Desenvolvimento de *FPGA DE1-SoC*, que incorpora o *Cyclone V SoC*, o qual, por sua vez, integra o processador Cortex-A9; a fonte 12V-2A para alimentação do DE1-SoC; o cartão microSDHC para armazenamento de dados; e o roteador Keo KLR 300N, responsável pela comunicação de rede entre o computador de implementação e o kit de desenvolvimento DE1-SoC durante os testes. Além disso, o Raspberry Pi 5 foi utilizado como parte integrante do estudo de caso, complementando o ambiente experimental. São necessários também cabos USB do tipo A para tipo B e do tipo A para tipo Mini B para as conexões entre os equipamentos.

3.1.2 Softwares

Esta seção descreve, em mais detalhes, os programas necessários para o desenvolvimento do estudo de caso e avaliação das métricas.

Quartus II

O *Quartus II* é o ambiente de desenvolvimento integrado (IDE) da Intel (anteriormente Altera) utilizado para a síntese e programação de dispositivos FPGA, como o DE1-SoC. No contexto deste experimento, o *Quartus II* foi utilizado para implementar a

lógica de hardware dos blocos de processamento digital de imagem (PDI), que serão melhor descritos na seção 3.3; configurar e mapear as conexões de entrada e saída internas e externas para integração com outros módulos de hardware, como o Raspberry Pi e o HPS; realizar simulações e testes dos blocos; e gerar os arquivos de programação (bitstream) para configuração do FPGA, que posteriormente foram carregados no dispositivo para realizar o experimento.

Altera SoC EDS

O SoC EDS (Embedded Design Suite) é um conjunto de ferramentas voltado ao desenvolvimento de sistemas embarcados, especialmente SoCs, que combinam FPGA e processadores ARM, como o Cyclone V. No contexto deste experimento, o SoC EDS foi utilizado para facilitar a configuração do sistema operacional e dos dispositivos no HPS, permitindo a execução de código no processador ARM do Cyclone V e, consequentemente, a comunicação com o FPGA para tarefas de controle e processamento. Além disso, o SoC EDS foi empregado para agrupar os elementos lógicos em bibliotecas, como endereços de registradores, valores de deslocamento de memória e outras constantes definidas pela Altera.

Linux Yocto

O *Yocto* é um projeto que possibilita a criação de distribuições Linux personalizadas para sistemas embarcados, sendo empregado no SoC Cyclone V, especificamente no ARM Cortex-A9, para fornecer o ambiente de execução do experimento. As principais atividades realizadas com o Linux Yocto incluem a configuração e compilação do sistema operacional embarcado, garantindo suporte a periféricos essenciais como Ethernet e USB, além das interfaces e serviços associados a protocolos como *SSH*, *DHCP* e *DNS*. Também foi feita a integração com bibliotecas de baixo nível, permitindo o acesso direto ao hardware do SoC e a execução de operações em tempo real, como a captura e análise de dados de imagem.

GNU Compiler Collection

O **GCC**, especificamente a versão *arm-none-linux-gnueabihf-gcc*, foi o compilador utilizado para compilar o código-fonte em C/C++ destinado ao processador ARM do Cyclone V e ao sistema Linux Yocto. Com o GCC, foi realizada a compilação do código responsável pelo controle das operações de comunicação SPI e pelos blocos de reconhecimento de gestos, gerando binários compatíveis com o processador ARM embarcado. O código também foi otimizado para reduzir o tempo de execução e melhorar o desempenho do sistema, especialmente nas funções de pré-processamento de imagem e controle de comunicação entre o FPGA e o Raspberry Pi. Além disso, o GCC foi utilizado para gerar arquivos executáveis e bibliotecas para o SoC, que foram integrados ao sistema ope-

racional embarcado e executados manualmente após a transferência para o Linux Yocto. Foram realizados testes de compilação cruzada, permitindo que o código fosse compilado em um ambiente de desenvolvimento (Windows) e transferido para execução no sistema embarcado (Linux ARM).

Visual Studio Code

O **Visual Studio Code** foi utilizado como ambiente de desenvolvimento principal para edição, organização e documentação do código-fonte do experimento. Com o Visual Studio Code, foram realizadas atividades como a edição do código-fonte em Python, C/C++ e scripts utilitários para compilação, integrando ferramentas de depuração e controle de versão (Git). Também foram utilizadas extensões para destacar sintaxe, gerenciar dependências de bibliotecas e verificar a conformidade do código com os padrões do GCC e do sistema embarcado. O ambiente foi ainda empregado na implementação e teste de scripts de automação para a transferência de arquivos e sincronização do código entre o ambiente de desenvolvimento e o SoC Cyclone V, executada via SSH.

3.2 Configuração do ambiente

Nesta seção, são descritos os componentes físicos e o sistema operacional configurados para o estudo de caso de reconhecimento de gestos, destacando o papel de cada um no sistema.

3.2.1 FPGA DE1-SoC

O kit FPGA DE1-SoC foi a base da arquitetura do estudo de reconhecimento de gestos. É integrado com um processador ARM Cortex-A9 dual-core de até 925 MHz. O sistema foi configurado utilizando o Quartus II, com componentes sintetizados para o processamento da imagem e aquisição das features, além do uso do HPS (Cortex-A9). Foi conectado ao computador de desenvolvimento utilizando os cabos USB previamente descritos e se tornou comunicável via rede por meio de sua porta Ethernet, utilizando um roteador, como mostra a figura 3.1. A programação, suporte de bibliotecas, funções e constantes necessárias foram fornecidos pela suíte de ferramentas Altera SoC EDS, que auxilia na configuração do HPS e na interação com o FPGA.

Figura 3.1: Conjunto DE1-SoC ligado ao roteador KLR 300N

Fonte: Autoria própria

3.2.2 Cartão microSDHC

O cartão microSDHC foi empregado para armazenar a imagem do sistema operacional Yocto e seu sistema de arquivos, sendo conectado ao DE1-SoC no conector de cartão microSD, conforme mostrado na Figura 3.2. Com 16 GB de capacidade total, o cartão utilizou apenas 1,1 GB para os arquivos do sistema. O Yocto, um sistema operacional baseado em Linux, será abordado em detalhes em seções posteriores. A Figura 3.2 ilustra a posição do cartão microSDHC no DE1-SoC.

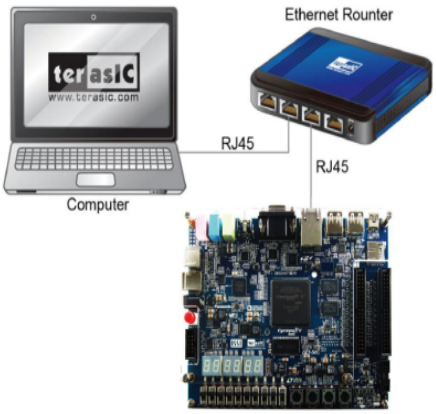
Figura 3.2: Conjunto microSDHC e DE1-SoC

Fonte: Terasic Inc. (2019b)

3.2.3 Roteador Keo KLR 300N

O roteador Keo KLR 300N foi utilizado em modo repetidor *wireless* para estabelecer a comunicação entre o computador de desenvolvimento, que implementa e envia o código, e o sistema operacional executado no ARM Cortex-A9 do DE1-SoC. O roteador possui 3 portas LAN e uma porta WAN. A porta LAN 1 foi conectada ao DE1-SoC, conforme ilustrado na figura 3.3, permitindo a comunicação eficiente entre os dispositivos.

Figura 3.3: Cabo Ethernet conectado à DE1-SoC



Fonte: Terasic Inc. (2019b)

3.2.4 Raspberry Pi 5

A Raspberry Pi 5, equipada com o sistema operacional Raspbian, foi utilizada como uma hipótese alternativa ao HPS integrado no DE1-SoC, atuando como plataforma externa para comunicação e controle do FPGA. Um *script* em Python, desenvolvido pelo autor e disponibilizado em Nascimento (2024b), foi empregado para estabelecer a comunicação entre os dispositivos, permitindo a Raspberry Pi solicitar o processamento de imagens e a extração de features no DE1-SoC, enviando comandos e recebendo respostas. Além disso, a Raspberry Pi 5 também foi utilizada para executar de forma independente toda a *pipeline* de PDI, processo que será detalhado nas seções seguintes.

3.3 Reconhecimento de gestos

Nesta seção, são detalhadas as etapas de processamento de imagem digital (PDI), definida com base no modelo proposto por Gupta et al. (2012) para reconhecimento de gestos em tempo real utilizando o kit de desenvolvimento DE1-SoC. Além disso, serão definidas as arquiteturas elaboradas para o estudo de caso, isto é Raspberry Pi 5 (Externa) — Cyclone V (DE1-SoC), Cortex-A9 (DE1-SoC) — Cyclone V (DE1-SoC), bem como o protocolo de comunicação utilizado nas arquiteturas que utilizam a FPGA Cyclone V. O código-fonte necessário para a síntese dos componentes presentes na FPGA está disponibilizado em Nascimento (2024c).

3.3.1 Gestos selecionados para análise

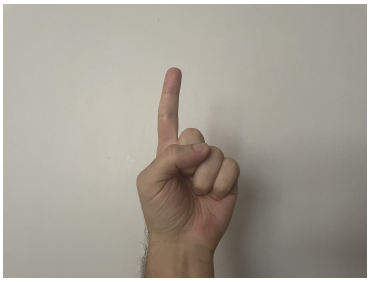
Neste estudo, foi analisado um subconjunto dos gestos definidos por Gupta et al. (2012). Dos 10 gestos originalmente propostos, foram selecionados 6, que foram processados e avaliados utilizando uma *pipeline* implementada na DE1-SoC. A escolha desses gestos considerou critérios de representatividade, priorizando padrões variados para testar e validar diferentes arquiteturas. A descrição detalhada da *pipeline* utilizada será apresentada em uma seção posterior deste trabalho. As figuras a seguir ilustram os gestos selecionados:

Figura 3.4: Gestos analisados no estudo, representando diferentes padrões utilizados na *pipeline* de processamento

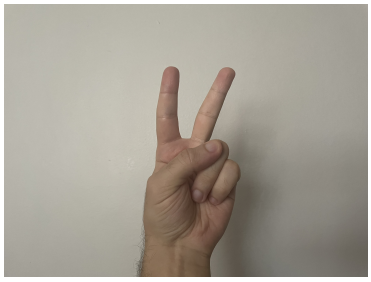
(a) Gesto 1: Palma da mão aberta

(b) Gesto 2: Punho fechado

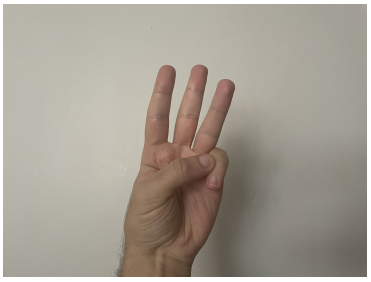
(c) Gesto 3: Um dedo erguido



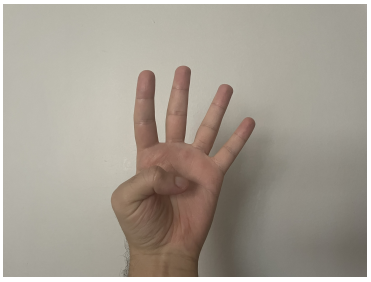
(d) Gesto 4: V de vitória



(e) Gesto 5: Três dedos erguidos



(f) Gesto 6: Quatro dedos erguidos

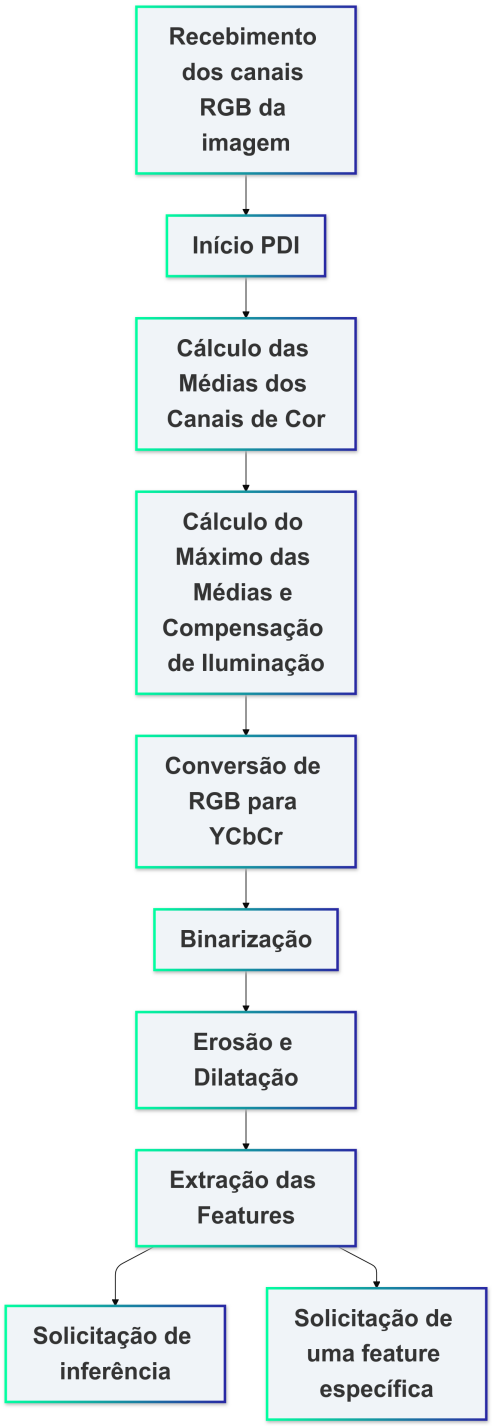


Fonte: Contribuição pessoal de Melo (2024)

3.3.2 Definição das etapas da *pipeline*

O processo de reconhecimento de gestos segue uma *pipeline* estruturada, composta por várias etapas sequenciais de processamento ilustradas na figura 3.5, com base no modelo descrito por Gupta et al. (2012). Esse fluxo de processamento culmina na extração das *features* da imagem, essenciais para a identificação dos gestos. Cada etapa da *pipeline* é descrita nos tópicos abaixo.

Figura 3.5: Etapas executadas até extração das features



Fonte: Autoria própria

Cálculo das médias dos canais de cor

Nesta primeira etapa, calculam-se as médias dos píxeis de cada canal de cor (vermelho, verde e azul) da imagem (que, por sua vez, possui 320 píxeis de largura e 240 píxeis de altura), conforme descrito no modelo de Gupta et al. (2012). Esses valores são utili-

zados para uma análise preliminar da imagem e como base para o ajuste de iluminação. Os novos valores das médias armazenados para os canais de cor vermelho, verde e azul (R_{medio}, G_{medio} e B_{medio}), são descritos pelas equações (3.1), (3.2) e (3.3), respectivamente. Onde $I(x, y, k)$ é um píxel da imagem nas coordenadas (x, y) do canal k, onde k=1,2,3 ou vermelho, verde e azul, respectivamente.

$$R_{medio} = \frac{\sum_{x=1}^{240} \sum_{y=1}^{320} I(x, y, 1)}{320 \times 240} \tag{3.1}$$

$$G_{medio} = \frac{\sum_{x=1}^{240} \sum_{y=1}^{320} I(x, y, 2)}{320 \times 240} \tag{3.2}$$

$$B_{medio} = \frac{\sum_{x=1}^{240} \sum_{y=1}^{320} I(x, y, 3)}{320 \times 240} \tag{3.3}$$

Cálculo do máximo das médias e compensação de iluminação

Com as médias calculadas, determina-se o valor máximo entre os canais (ϵ), conforme descrito na equação (3.4). Esse valor é utilizado para realizar a compensação de iluminação, ajustando a intensidade da imagem e reduzindo variações, como proposto por Gupta et al. (2012). Os ajustes para os canais vermelho, verde e azul são dados, respectivamente, pelas equações (3.5), (3.6) e (3.7). Onde $R(x, y)$, $G(x, y)$ e $B(x, y)$ representam, respectivamente, os valores dos píxeis nas coordenadas (x, y) para os canais vermelho, verde e azul da imagem.

$$\epsilon = \max(R_{medio}, G_{medio}, B_{medio}) \tag{3.4}$$

$$R(x, y) = \frac{R(x, y) \cdot R_{medio}}{\epsilon} \tag{3.5}$$

$$G(x, y) = \frac{G(x, y) \cdot G_{medio}}{\epsilon} \tag{3.6}$$

$$B(x, y) = \frac{B(x, y) \cdot B_{medio}}{\epsilon} \tag{3.7}$$

Conversão de espaço de cor de RGB para YCbCr

Após a compensação de iluminação, a imagem é convertida do espaço de cor RGB para o espaço YCbCr, seguindo o modelo de Gupta et al. (2012). Esse espaço de cor é amplamente utilizado em tarefas de processamento e reconhecimento de imagem devido ao seu desempenho em aprimoramento, compressão e na representação de frequências, como também indicado por Rehman et al. (2024). Essa conversão facilita a separação

entre o objeto (mão) e o fundo, sendo uma representação mais adequada para operações de segmentação por cor. A conversão dos canais RGB para o espaço YCbCr pode ser dada pelas seguintes equações:

$$Y(x,y) = 0,299 \cdot R(x,y) + 0.587 \cdot G(x,y) + 0,114 \cdot B(x,y) \tag{3.8}$$

$$C_B(x,y) = 128 - 0,168736 \cdot R(x,y) - 0,331264 \cdot G(x,y) + 0,5 \cdot B(x,y) \tag{3.9}$$

$$C_R(x,y) = 128 + 0,5 \cdot R(x,y) - 0,418688 \cdot G(x,y) - 0,081312 \cdot B(x,y) \tag{3.10}$$

Apesar disso, neste trabalho, não foi considerada a contribuição do canal de luminância (Y) para a segmentação da imagem, uma vez que focar nos canais C_B e C_R permite uma segmentação mais precisa, pois eles carregam a informação essencial sobre a cor, independentemente das variações de iluminação que influenciam o canal Y.

Binarização

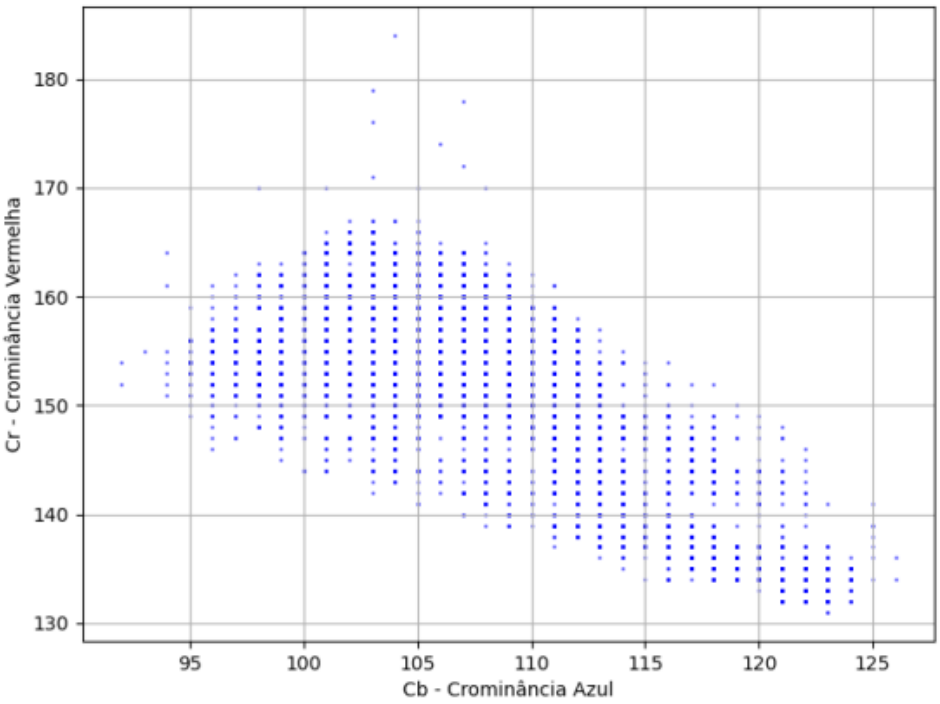
A imagem em YCbCr é então binarizada, isto é, convertida para pixeis de cor preta ou branca. Esta etapa segmenta a mão do fundo, gerando uma máscara binária que simplifica as operações subsequentes, conforme o modelo referenciado. Os resultados da segmentação podem ser vistos na Figura 3.8b. Ao fim, a imagem mantém as dimensões de altura e largura, mas agora possui somente um canal. Neste estudo, os valores para um píxel (*byte*), considerados limiares para a segmentação, foram de:

$$90 \leq C_B \leq 120$$

$$140 \leq C_R \leq 170$$

Os valores apresentados foram obtidos a partir do gráfico 3.6 de C_R *contra* C_B, que representa a dispersão dos canais de cores para um dos gestos (ver Figura 3.4a) previamente selecionados. Para definir os limiares, foi realizada uma análise da região do gráfico correspondente à mão, em contraste com a região que representa o fundo da imagem.

Figura 3.6: Gráfico de dispersão C_R versus C_B



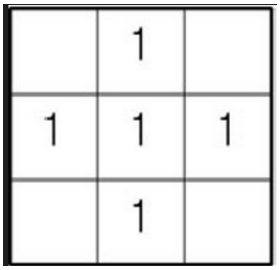
Fonte: Autoria própria

Erosão e dilatação

Tendo em vista a necessidade do aperfeiçoamento da imagem, aplicam-se as operações morfológicas de erosão e dilatação, conforme o fluxo descrito por Gupta et al. (2012), para eliminar ruídos e aprimorar os contornos da mão na imagem, a operação resultante está descrita na equação 3.11, onde A representa a imagem ao qual incidirá a filtragem e B o elemento estruturante, ilustrado na Figura 3.7.

$$A \circ B = (A \ominus B) \oplus B \tag{3.11}$$

Figura 3.7: Elemento estruturante da filtragem

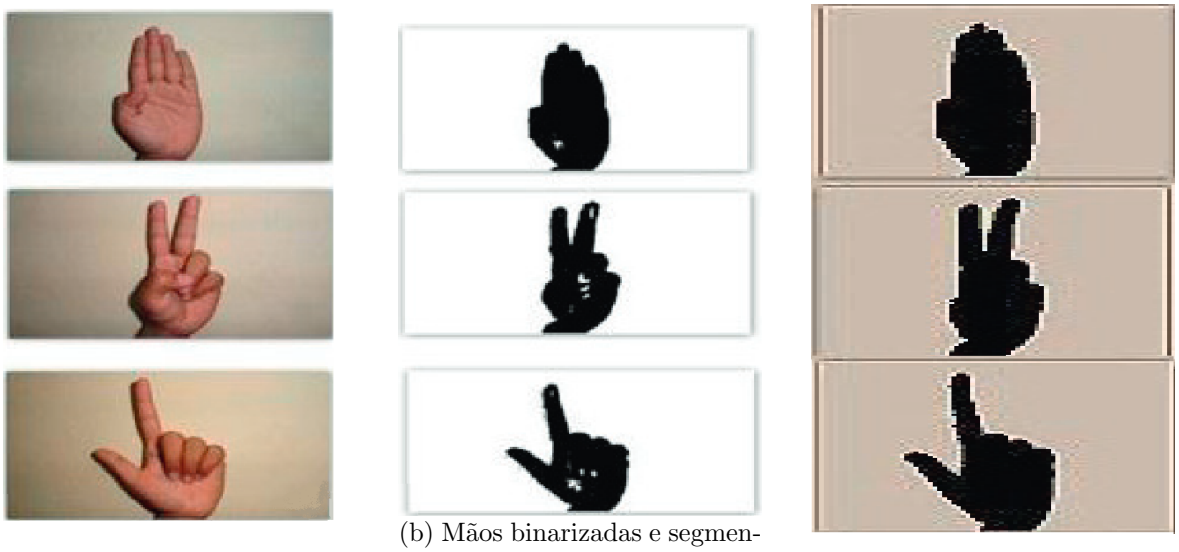


Fonte: Autoria própria

A erosão remove pequenos pontos de ruído, enquanto a dilatação reforça as bordas

do objeto segmentado. O resultado dessas operações, bem como a transição desde a imagem original até o fim do PDI executado na *pipeline*, podem ser dados pelas imagens presentes na figura 3.8:

Figura 3.8: Resultado das operações de segmentação e filtragem morfológica



(a) Imagens originais

(b) Mãos binarizadas e segmentadas

(c) Imagem filtrada

Fonte: Adaptado de Gupta et al. (2012)

Extração das features para o reconhecimento de gestos

Na etapa final da *pipeline*, são extraídas as *features* mais relevantes da imagem binarizada, como a área e o perímetro da mão, além da quantidade de dedos erguidos. Essas *features* são fundamentais para o processo de reconhecimento gestual, conforme a metodologia descrita por Gupta et al. (2012). Cada uma dessas características será explorada em detalhe nos tópicos subsequentes.

Área da mão:

A área da mão é calculada a partir da quantidade de píxeis que compõem a região da mão na imagem binarizada, utiliza-se, portanto, a equação 3.12. Esta *feature* fornece informações sobre o tamanho relativo da mão e é essencial para distinguir gestos baseados no espaço ocupado pela mão. Onde $I_{filtrada}(x, y)$ representam os valores dos píxeis da imagem resultante dos processos descritos em 3.3.2 nas coordenadas (x, y) para os canais vermelho, verde e azul da imagem.

$$\text{Area} = \sum_{x=1}^{240} \sum_{y=1}^{320} I_{filtrada}(x, y) \tag{3.12}$$

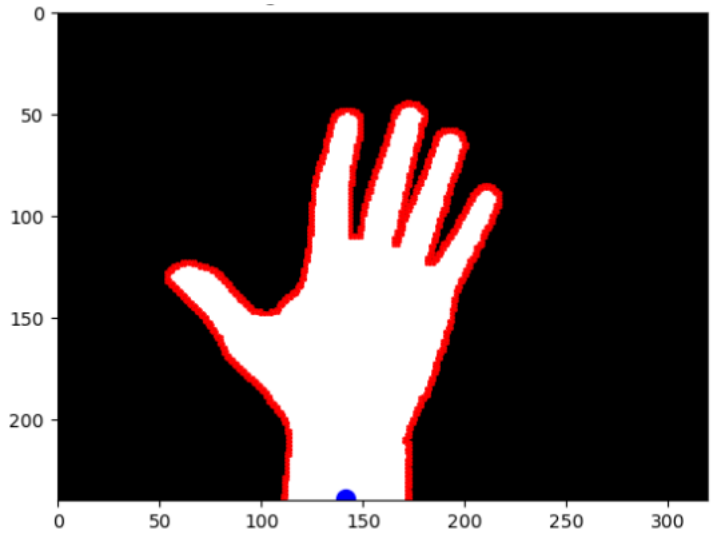
Perímetro da mão:

O perímetro da mão é determinado pela contagem dos píxeis ao longo da borda da mão. A operação pode ser descrita conforme a equação 3.13, na qual a função $Edge(I_{filtrada}(x, y))$ descreve a operação que identifica as bordas da mão, com base na transição de píxeis preto para branco e vice-versa, performada em todos os píxeis da imagem ao fim dos processos citados em 3.3.2 para cada ponto das coordenadas (x, y). Esta *feature* é útil para identificar a forma da mão e é particularmente importante para gestos no qual o contorno da mão, assim como na figura 3.9, é uma característica distintiva.

$$\text{Per} = \sum_{x=1}^{240} \sum_{y=1}^{320} Edge(I_{filtrada}(x, y)) \tag{3.13}$$

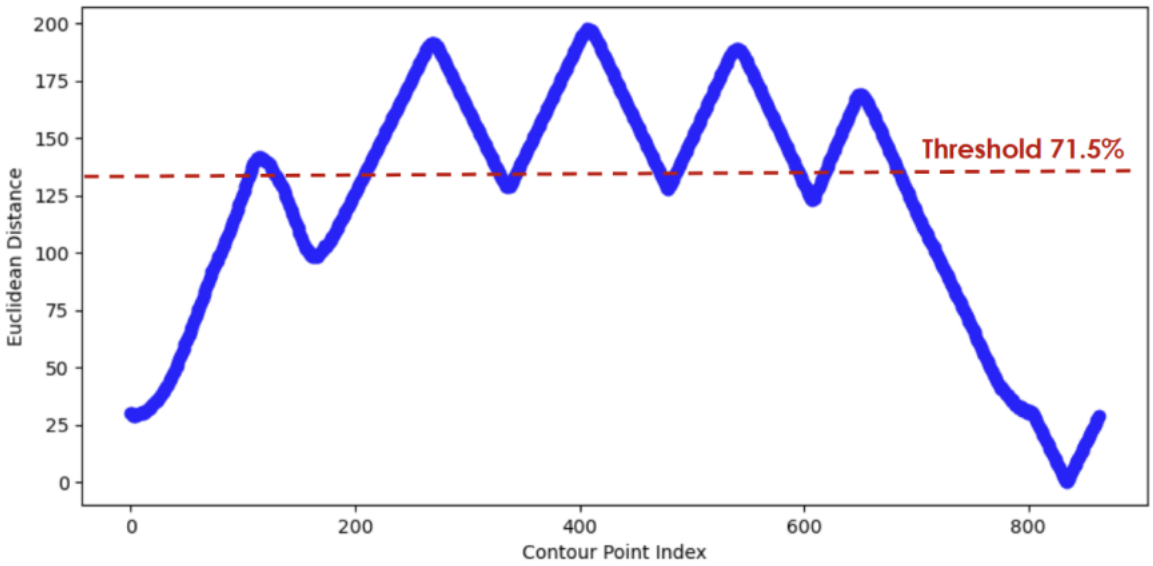
Quantidade de dedos erguidos: Essa *feature* determina o número de dedos visíveis na imagem, a partir do perfil radial dos píxeis do contorno da mão. Cada píxel do contorno tem uma distância euclidiana medida a partir de um ponto central no punho da mão, como ilustrado na figura 3.9. Os dedos erguidos podem ser identificados como os picos superiores nos perfis radiais, conforme mostrado na figura 3.10. Para melhorar a precisão na contagem, é aplicado uma etapa de filtragem, onde são descartados todos os picos cujos valores são inferiores a 71,5% do valor do maior pico observado. Esse filtro visa eliminar picos menores que poderiam ser erroneamente contados como dedos levantados, resultando em uma contagem mais precisa dos dedos visíveis.

Figura 3.9: Pixeis do contorno (vermelhos) e central do punho (azul) do gesto 3.4a



Fonte: Autoria própria

Figura 3.10: Perfis radiais dos píxeis do contorno da mão, no gesto 3.4a



Fonte: Autoria própria

Classificação do gesto

Uma vez obtidas as features foi possível classificar os gestos informados na imagem. A tabela 3.1 explicita como foi feita a classificação dos gestos, quais *features* foram utilizadas e de qual modo. Ao fim do processamento, as *features* são armazenadas e disponibilizadas para solicitação do dispositivo controlador, por meio dos pacotes definidos em 3.4.1.

Tabela 3.1: Classificação dos gestos baseada nas features de perímetro da mão e quantidade de dedos erguidos.

Gesto	**Perímetro da Mão**	**Quantidade de Dedos**
Gesto 3.4c	440 a 660	1
Gesto 3.4d	440 a 660	2
Gesto 3.4e	440 a 660	3
Gesto 3.4f	> 610	4
Gesto 3.4a	≥ 687	5
Gesto 3.4b	< 381	Independe
Indefinido	Exceção para casos acima	Exceção para casos acima

Fonte: Autoria própria

Como descrito na tabela 3.1 e diferente do modelo proposto por Gupta et al. (2012), este estudo não utilizou a área da mão como parâmetro para avaliação dos gestos. Durante os testes iniciais, para o subconjunto de gestos selecionados, não foi identificada uma diferença na inferência do modelo com relação ao uso da área da mão como *feature*. Além

disso, constatou-se um aumento substancial no tempo de síntese dos componentes de PDI, em razão da necessidade de comparar três variáveis simultaneamente (área, perímetro e número de dedos). Em função dessas observações, optou-se por excluir a contribuição da *feature* de área da mão no processo de inferência.

3.4 Arquitetura e interconexão dos componentes

Esta seção aborda a estrutura e os processos de comunicação entre os diferentes componentes do sistema, destacando a implementação das arquiteturas envolvidas e os protocolos de comunicação utilizados. São discutidas as interações entre a Raspberry Pi 5, o FPGA Cyclone V e o Cortex-A9, bem como a definição e a implementação do protocolo SPI, crucial para a troca de dados na *pipeline* de Processamento Digital de Imagens (PDI). A integração entre essas plataformas e a escolha das tecnologias de comunicação são descritas em detalhes nos itens seguintes.

3.4.1 Protocolo de comunicação SPI

O protocolo SPI foi escolhido para viabilizar a comunicação entre o dispositivo controlador e o periférico. Com base nele, foi desenvolvido um protocolo de comunicação específico, definindo a estrutura de pacotes e cabeçalhos para a troca de dados. No contexto do reconhecimento de gestos, os dados são transmitidos *byte* a *byte* no modo *MSB first*, priorizando o envio do *bit* mais significativo. As operações e estados dessa comunicação personalizada são apresentados nos tópicos a seguir:

Sem operação, execução normal da comunicação, estado de pronto:
Esse estado (indicado na tabela 3.2) representa a situação em que a FPGA está inativa e pronta para receber comandos. Nenhuma operação está em execução e não há retorno ao dispositivo controlador.

Tabela 3.2: Sem retorno ao dispositivo controlador, FPGA aguardando solicitações

Bit	7	6	5	4	3	2	1	0
Valor	0	0	-				-	

Fonte: Autoria própria

PDI em execução:
Indica, conforme a tabela 3.3, que a FPGA iniciou a *pipeline* de Processamento Digital de Imagem (PDI). A comunicação com o dispositivo controlador continua ativa, mas a prioridade é o processamento interno.

Tabela 3.3: Cabeçalho de execução de PDI na FPGA

Bit	7	6	5	4	3	2	1	0
Valor	0	1	-				-	

Fonte: Autoria própria

Envio de imagem — Canal Vermelho:
Cabeçalho utilizado para identificar a transferência dos dados relacionados ao canal vermelho de uma imagem, indicado na tabela 3.4. Esse comando é emitido pelo dispositivo controlador para iniciar a transmissão correspondente.

Tabela 3.4: Cabeçalho de solicitação de envio de canal vermelho da imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	0	0	1	0	1

Fonte: Autoria própria

Envio de imagem — Canal Verde:
Similar ao canal vermelho, este cabeçalho é usado para o envio de dados do canal verde da imagem, conforme indicado na tabela 3.5, permitindo sua individualização no processamento.

Tabela 3.5: Cabeçalho de solicitação de envio de canal verde da imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	0	0	1	1	0

Fonte: Autoria própria

Envio de imagem — Canal Azul:
Comando responsável por solicitar à FPGA a transmissão do canal azul de uma imagem, indicado na tabela 3.6. Cada canal pode ser processado separadamente para operações específicas.

Tabela 3.6: Cabeçalho de solicitação de envio de canal azul da imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	0	0	1	1	1

Fonte: Autoria própria

Solicitação de imagem — Canal Vermelho:
O dispositivo controlador requisita à FPGA, conforme indicado na tabela 3.7, o envio dos

dados correspondentes ao canal vermelho de uma imagem previamente processada pela *pipeline* ou armazenada.

Tabela 3.7: Cabeçalho de solicitação de recebimento de canal vermelho da imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	0	1	0	0	1

Fonte: Autoria própria

Solicitação de imagem — Canal Verde:
Esse cabeçalho indica uma solicitação para receber o canal verde da imagem processada ou armazenada na FPGA, conforme indicado na tabela 3.8.

Tabela 3.8: Cabeçalho de solicitação de recebimento de canal verde da imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	0	1	0	1	0

Fonte: Autoria própria

Solicitação de imagem — Canal Azul:
Comando utilizado para requisitar o canal azul da imagem, indicado na tabela 3.9, possibilitando a composição completa da imagem RGB pelo dispositivo controlador.

Tabela 3.9: Cabeçalho de solicitação de recebimento de canal azul da imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	0	1	0	1	1

Fonte: Autoria própria

Solicitação de início do PDI:
Comando enviado pelo dispositivo controlador para iniciar o processo de análise de imagem na FPGA, ativando o módulo de Processamento Digital de Imagens (PDI). Este cabeçalho marca o início da sequência de operações, conforme indicado na tabela 3.10.

Tabela 3.10: Cabeçalho de solicitação de processamento do PDI

Bit	7	6	5	4	3	2	1	0
Valor	-		0	0	1	1	-	

Fonte: Autoria própria

Solicitação de reconhecimento de gesto:
Cabeçalho destinado à requisição de inferência sobre gestos detectados em uma imagem

processada. O dispositivo controlador solicita, conforme indicado na tabela 3.11, que a FPGA realize a identificação com base nos dados capturados.

Tabela 3.11: Cabeçalho de solicitação de inferência do gesto detectado na imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	1	1	1	-	

Fonte: Autoria própria

Solicitação de área da mão calculada:
Utilizado para requisitar à FPGA o valor da área total da mão identificada na imagem processada, conforme indicado na tabela 3.12.

Tabela 3.12: Cabeçalho de solicitação de inferência do gesto detectado na imagem

Bit	7	6	5	4	3	2	1	0
Valor	-		0	1	0	0	-	

Fonte: Autoria própria

Solicitação de perímetro da mão calculado:
Esse cabeçalho sinaliza a solicitação para cálculo do perímetro da mão detectada na imagem, indicado na tabela 3.13 abaixo.

Tabela 3.13: Cabeçalho de solicitação de cálculo do perímetro da mão detectado

Bit	7	6	5	4	3	2	1	0
Valor	-		0	1	0	1	-	

Fonte: Autoria própria

Solicitação de quantidade de picos da mão detectados:
Comando responsável por requisitar o número de picos, ou seja, os dedos erguidos identificados na mão detectada, conforme indicado na tabela 3.14.

Tabela 3.14: Cabeçalho de solicitação de número de dedos erguidos

Bit	7	6	5	4	3	2	1	0
Valor	-		0	1	1	0	-	

Fonte: Autoria própria

Requisições de envio de imagem:
Quando o cabeçalho indicar o envio de dados de algum canal da imagem, é obrigatório

informar informações adicionais para configurar a transferência. Os dois *bytes* subsequentes ao cabeçalho devem conter a altura da imagem, expressa em *bytes* (ou seja, o número de linhas de píxeis). Em seguida, nos dois *bytes* seguintes, deve-se especificar a largura da imagem, também em *bytes* (o número de colunas de píxeis). Após essa configuração, os próximos k *bytes* transmitidos devem representar os dados dos píxeis da imagem. Esses píxeis devem estar organizados em um vetor unidimensional, resultado do achatamento da imagem bidimensional (*Flatten Image*), onde o valor de k é calculado como $k = altura \times largura$ da imagem.

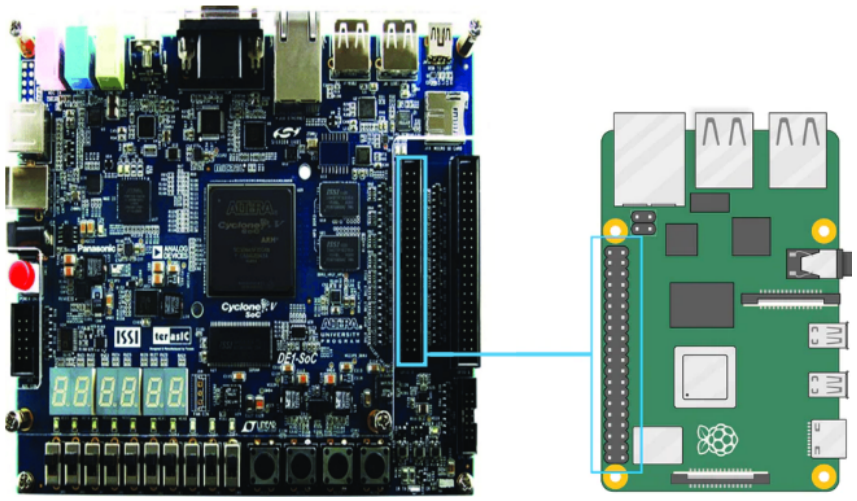
3.4.2 Arquitetura Raspberry Pi 5 desacoplada do FPGA

Nesta arquitetura, a Raspberry Pi 5 foi utilizada isoladamente, com o objetivo de validar a hipótese de que a execução da *pipeline* de PDI diretamente no dispositivo poderia ser uma alternativa vantajosa em comparação ao processamento colaborativo com *FPGAs*. Utilizando o *OpenCV*, a Raspberry Pi efetuava a leitura de uma das imagens propostas em 3.3 diretamente do diretório onde o *script* em Python (disponível em Nascimento (2024b)) estava. Uma vez feita a leitura, a Raspberry Pi 5 pôde realizar as operações de processamento de imagens e extração de *features* diretamente em sua arquitetura, reproduzindo a mesma funcionalidade da *pipeline* originalmente desenvolvida para o DE1-SoC.

3.4.3 Arquitetura Raspberry Pi 5 - Cyclone V (DE1-SoC)

A arquitetura proposta nesta seção descreve a integração entre a Raspberry Pi 5 e a *pipeline* de Processamento Digital de Imagens (PDI) implementada na *FPGA Cyclone V* (DE1-SoC). Para realizar a comunicação entre esses dois componentes, utilizou-se o protocolo SPI (Serial Peripheral Interface), aproveitando os pinos SPI disponíveis na Raspberry Pi 5, conforme figuras 3.11 e 3.12.

Figura 3.11: Esquema de conexão entre a Raspberry Pi 5 e a FPGA Cyclone V (DE1-SoC) via SPI.



Fonte: Autoria própria

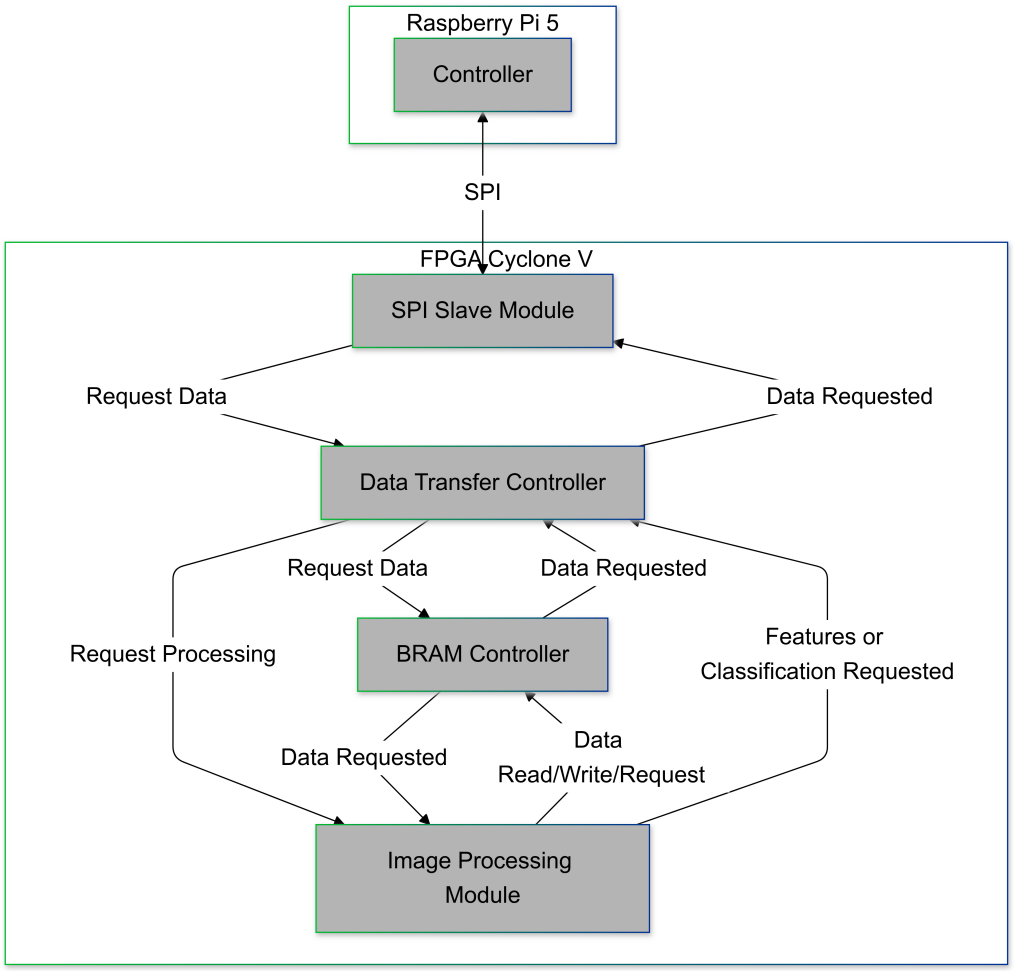
Figura 3.12: Pinos utilizados para o SPI da Raspberry Pi 5 e FPGA Cyclone V (DE1-SoC)

PIN	RPi5	FPGA
MOSI	GPIO 10	GPIO_0_D1
MISO	GPIO 9	GPIO_0_D0
SCK	GPIO 11	GPIO_0_D2
SS	GPIO 7	GPIO_0_D3
GND	PIN 20	PIN 12

Fonte: Autoria própria

A Raspberry Pi 5 foi configurada para atuar como dispositivo controlador, enquanto a FPGA, que contém a *pipeline* de PDI, atuou como dispositivo periférico. O diagrama presente na figura 3.13 ilustra o funcionamento dos elementos supracitados em conjunto.

Figura 3.13: Esquema de conexão entre a Raspberry Pi 5 e a FPGA Cyclone V (DE1-SoC) via SPI.



Fonte: Autoria própria

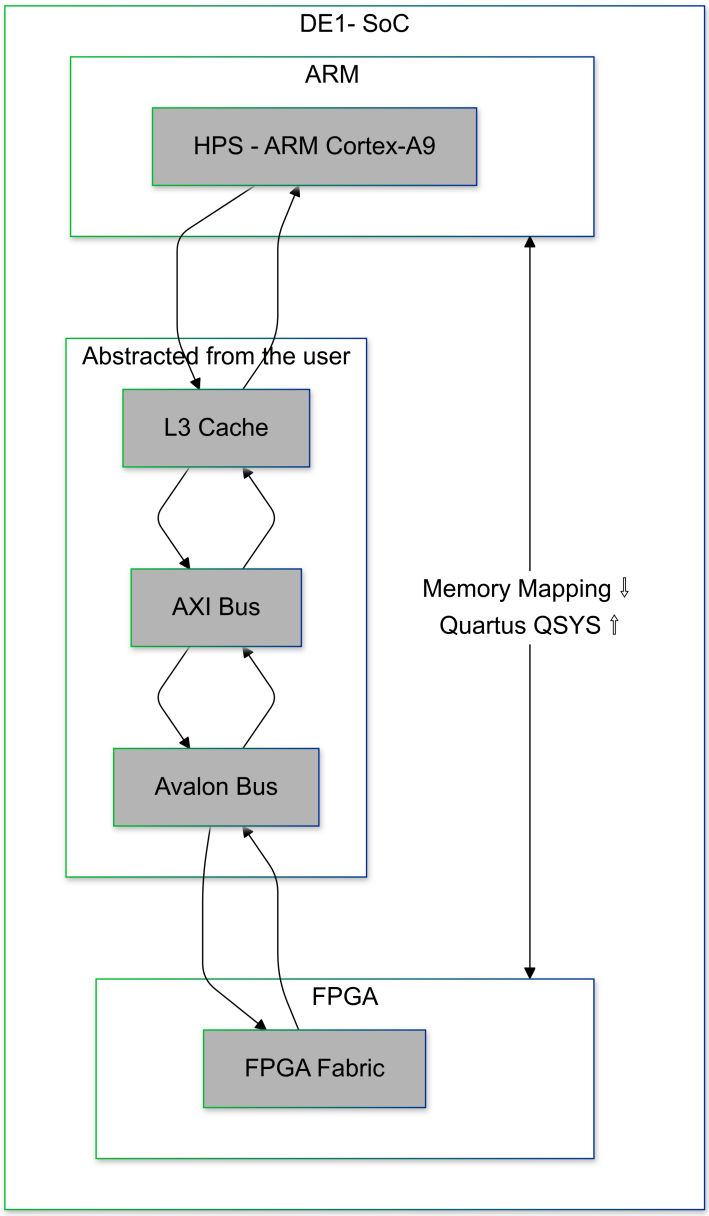
O sistema operacional Raspbian foi utilizado na Raspberry Pi 5 para gerenciar a comunicação via SPIDEV, facilitando a transmissão e recepção de dados SPI. O processo de comunicação consiste no envio de pacotes de dados da Raspberry Pi 5 para a FPGA, onde são processados pela *pipeline* de PDI. Após o processamento, os resultados são retornados a Raspberry Pi para análise posterior. Esse fluxo contínuo de dados permite a operação integrada entre os dispositivos, com a Raspberry Pi atuando como controlador e a FPGA executando o processamento em tempo real. Todas as operações citadas previamente e seus respectivos códigos-fontes encontram-se no repositório disponibilizado em Nascimento (2024b).

3.4.4 Arquitetura Cortex-A9 integrado à FPGA Cyclone V

O Cortex-A9, integrado ao SoC Cyclone V, foi configurado para atuar como controlador no sistema, executando o sistema operacional Linux Yocto, cuja imagem foi disponibilizada pela *Terasic*, especificamente para a DE1-SoC. Nesse arranjo, o HPS comunica-se

diretamente com a FPGA Cyclone V por meio do protocolo SPI. Apesar da comunicação SPI ser mantida, ela ocorre internamente no próprio SoC, por meio de configurações realizadas no Quartus (via QSYS) e utilizando o barramento AXI para interligar o *HPS* e a *FPGA*. Essa integração reduz a necessidade de conexões físicas externas, uma vez que o *HPS* (utilizando o *SPI*), utiliza o *SPI Slave* presente como componente da FPGA e visível como periférico mapeado em memória. A figura 3.14 apresenta uma visão esquemática do barramento AXI utilizado para interconectar o HPS e a FPGA dentro do SoC.

Figura 3.14: Visualização do AXI Bus por parte do HPS e FPGA, por QSYS ou Memory Mapped IO

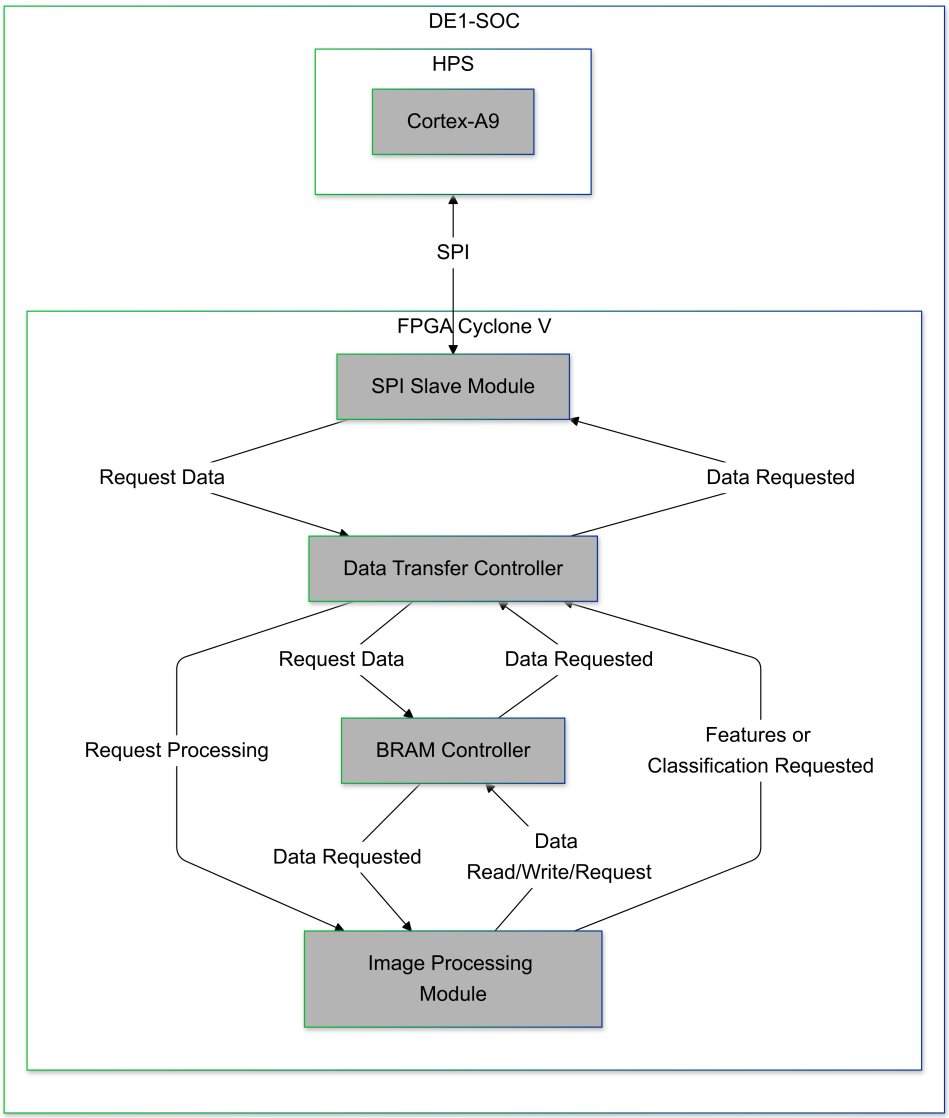


Fonte: Autoria própria

O Cortex-A9 realiza a coordenação do fluxo de dados e delega à FPGA o processamento paralelo dos dados provenientes da *pipeline* de PDI. A utilização do Linux

Yocto no Cortex-A9 viabilizou a configuração de drivers, a utilização da interface de rede e dos mecanismos de comunicação necessários para a troca de informações entre os dois componentes do SoC. O esquema presente na figura 3.15 demonstra o diagrama comportamental simplificado do Cortex-A9 e da FPGA, ambos operando como componentes internos integrados ao *SoC*. Com relação ao código-fonte necessário para a troca de dados entre Cortex-A9 e Cyclone V, o mesmo utilizou o próprio espaço de usuário, por meio de executáveis implementados previamente via *GCC* como descrito em 3.1.2. O repositório de código-fonte encontra-se disponibilizado por Nascimento (2024a).

Figura 3.15: Esquema de conexão entre o Cortex A9 integrado e a FPGA Cyclone V via SPI, ambos na DE1-SoC



Fonte: Autoria própria

Essa abordagem diferencia-se da arquitetura anterior, que utilizava a Raspberry Pi 5 conectada externamente. Na arquitetura com a Raspberry, a comunicação SPI ocorria

entre dispositivos separados, exigindo conexões físicas. Já no SoC, a comunicação SPI ocorre de forma intrínseca ao hardware, viabilizada pelo barramento *AXI*.

3.5 Métricas avaliadas

Neste trabalho, as métricas avaliadas focaram no desempenho do sistema em termos de tempo de processamento e comunicação. Especificamente, foram medidos o tempo total de envio da imagem, incluindo os três canais de cor (vermelho, verde e azul) combinados, e o tempo total de processamento, que abrange tanto o envio dos canais da imagem, quanto o processamento digital da imagem (PDI), a extração das features (área, perímetro e picos das mãos) e inferência do gesto conforme a seção 3.3.

Para analisar o desempenho individual das arquiteturas, também foram calculados os *throughputs*, que representam a quantidade de gestos processados por segundo. O cálculo foi feito a partir dos tempos totais de execução observados, conforme a equação 3.14:

$$\text{Throughput} = \frac{1}{T_{\text{Total}}(\text{segundos})} \tag{3.14}$$

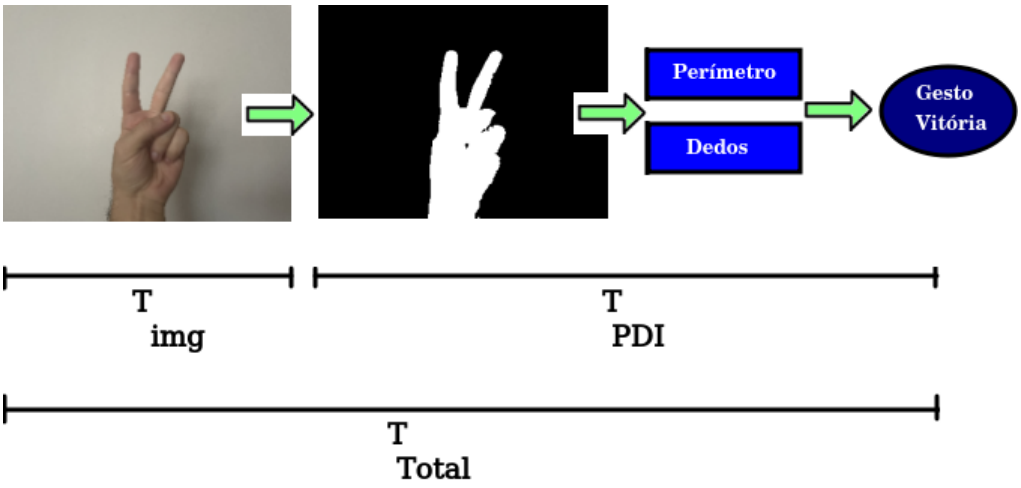
Onde T_{Total} é o tempo total de execução e inferência para cada gesto, conforme descrito previamente.

Capítulo 4

Resultados e Discussão

Neste capítulo, serão apresentados os resultados obtidos a partir dos testes realizados em três cenários: exclusivamente na Raspberry Pi 5, na arquitetura colaborativa entre Raspberry Pi 5 e FPGA Cyclone V, e na arquitetura integrada, utilizando o ARM Cortex-A9 e o Cyclone V do DE1-SoC. Para os dois casos das arquiteturas propostas no Capítulo 3 que utilizavam o SoC para efetuar processamento colaborativo, foram avaliados os tempos totais de envio dos canais da imagem combinados, processamento do gesto e recebimento da inferência, somadas às *features* extraídas do mesmo. Já para a Raspberry Pi 5 atuando de forma independente, não foi realizado o envio de dados para outro dispositivo, sendo considerado apenas o tempo necessário para o processamento digital de imagens (PDI) e a inferência diretamente no dispositivo. A figura 4.1 descreve os tempos adquiridos para avaliação e validação das arquiteturas.

Figura 4.1: Recapitulação da pipeline e seus respectivos tempos



Fonte: Autoria própria

4.1 Indicadores de desempenho das arquiteturas propostas

Esta seção apresenta e analisa os tempos de execução e o desempenho das diferentes arquiteturas avaliadas no experimento. São detalhadas as métricas obtidas para a Raspberry Pi 5 em execução independente, a integração entre a Raspberry Pi 5 e o DE1-SoC, e o processamento local no Cortex-A9 do DE1-SoC, permitindo compreender os impactos de cada abordagem no contexto do projeto.

4.1.1 Execução independente da Raspberry Pi 5

Primeiramente, na configuração onde a Raspberry Pi 5 executa exclusivamente o processamento da pipeline de PDI, sem interagir com o DE1-SoC, os tempos observados são relativos ao processamento local da imagem, sem necessidade de envio dos dados para dispositivos externos. O processo executado pela Raspberry Pi 5 foi previamente descrito na seção 3.4.2. O tempo total de processamento inclui apenas o tempo gasto pela Raspberry Pi 5 para processar a imagem e realizar as operações de extração de features, como mostrado na tabela 4.1.

Tabela 4.1: Tempos de processamento do PDI e Total, respectivamente, com relação ao uso exclusivamente da Raspberry Pi 5

Desempenho p/ Arquitetura		
Raspberry PI 5 (Exclusivamente)		
Gesto FPGA	$T_{PDI}(ms)$	$T_{Total}(ms)$
Palma da mão aberta (Figura 3.4a)	821,17	821,17
Punho fechado (Figura 3.4b)	746,76	746,76
Um dedo erguido (Figura 3.4c)	741,92	741,92
V de vitória (Figura 3.4d)	760,9	760,9
Três dedos erguidos (Figura 3.4e)	752,94	752,94
Quatro dedos erguidos (Figura 3.4f)	753,65	753,65

Fonte: Autoria própria

4.1.2 Integração da Raspberry Pi 5 com DE1-SoC:

A seguir, ao integrar a Raspberry Pi 5 ao DE1-SoC, por meio da Cyclone V, foi possível observar o impacto da comunicação entre os dispositivos. Nesta configuração, a Raspberry Pi 5 é responsável por enviar a imagem ao DE1-SoC, onde o processamento do gesto e a extração das features são realizados, retornando a inferência para a Raspberry Pi. O detalhamento das operações realizadas nessa arquitetura está descrito na seção

3.4.3, e os resultados dos testes executados podem ser consultados na Tabela 4.2.

Tabela 4.2: Tempos de envio da imagem, processamento do PDI e Total, respectivamente, com relação ao uso da Raspberry Pi 5 externa e FPGA Cyclone V

Desempenho p/ Arquitetura			
Raspberry PI 5 (Externa) - Cyclone V (DE1-SoC)			
Gesto FPGA	$T_{img}(ms)$	$T_{PDI}(ms)$	$T_{Total}(ms)$
Palma da mão aberta (Figura 3.4a)	89,172	32,238	121,41
Punho fechado (Figura 3.4b)	89,301	32,179	121,48
Um dedo erguido (Figura 3.4c)	89,448	32,192	121,64
V de vitória (Figura 3.4d)	89,391	32,199	121,59
Três dedos erguidos (Figura 3.4e)	89,446	32,224	121,67
Quatro dedos erguidos (Figura 3.4f)	89,325	32,625	121,95

Fonte: Autoria própria

4.1.3 Processamento integrado no DE1-SoC com o Cortex-A9 e Cyclone V

Na sequência, com a integração do ARM Cortex-A9 ao Cyclone V (DE1-SoC), todo o processamento do gesto, incluindo a extração das features, foi realizado localmente no SoC, eliminando a necessidade de comunicação com dispositivos externos. A seção 3.4.4 aborda em maior profundidade essa configuração, e os resultados experimentais dessa abordagem estão resumidos na Tabela 4.3.

Tabela 4.3: Tempos de envio da imagem, processamento do PDI e Total, respectivamente

Desempenho p/ Arquitetura			
Cortex-A9 (DE1-SoC) - Cyclone V (DE1-SoC)			
Gestos	$T_{img}(ms)$	$T_{PDI}(ms)$	$T_{Total}(ms)$
Palma da mão aberta (Figura 3.4a)	2.665,66	32,450	2.698,21
Punho fechado (Figura 3.4b)	2.666,39	32,340	2.698,84
Um dedo erguido (Figura 3.4c)	2.665,55	32,371	2.698,02
V de vitória (Figura 3.4d)	2.665,66	32,392	2.698,16
Três dedos erguidos (Figura 3.4e)	2.666,16	32,402	2.698,66
Quatro dedos erguidos (Figura 3.4f)	2.665,56	32,427	2.698,09

Fonte: Autoria própria

4.2 Análise do desempenho

Nesta seção, avaliam-se os tempos de execução e o desempenho de modo geral, das diferentes arquiteturas testadas no experimento. Serão discutidas as métricas obtidas para cada configuração, incluindo o uso exclusivo da Raspberry Pi 5, a integração entre a Raspberry Pi 5 e o DE1-SoC (Cyclone V), e o processamento local realizado pelo Cortex-A9 e Cyclone V no DE1-SoC. O objetivo é analisar os impactos de cada abordagem no desempenho global do sistema e identificar os pontos fortes e as limitações de cada arquitetura no contexto do projeto.

4.2.1 Comparação entre as arquiteturas

A comparação entre as arquiteturas propostas é baseada nos tempos de execução observados para cada configuração. Cada uma das abordagens – Raspberry Pi 5 isolada, Raspberry Pi 5 integrada ao DE1-SoC e processamento local no Cortex-A9 do DE1-SoC – oferece características distintas em termos de desempenho e eficiência.

Portanto, conforme os dados exibidos nas tabelas 4.1, 4.2 e 4.3, e segundo o teste de Kruskal-Wallis é possível afirmar que as diferenças entre os tempos para processamento do PDI e solicitação das features (T_{PDI}) são estatisticamente significativas e que as diferenças observadas são relevantes ($p-value = 0.001265$ e $\eta^2 = 0.76$, respectivamente). Além disso, por meio do teste t de Student, também pode-se dizer que a diferença dos tempos de envio da imagem nas arquiteturas descritas em 3.4.3 e 3.4.4 são estatisticamente significativos ($p - value < 0.00001$).

Para avaliar o desempenho individual das arquiteturas, foram calculados os *throughputs* em cada uma das configurações propostas. Ou seja, a quantidade de gestos processados por segundo para cada gesto, por arquitetura. Os resultados obtidos para cada arquitetura estão apresentados nas tabelas 4.4, 4.5 e 4.6.

Tabela 4.4: Throughputs calculados para cada gesto especificamente para a arquitetura Raspberry Pi 5

Raspberry Pi 5 (Sem Integração Externa)		
Gesto FPGA	$T_{Total}(ms)$	**Throughput (gesto/s)**
Palma da mão aberta (Figura 3.4a)	821,17	1,218
Punho fechado (Figura 3.4b)	746,76	1,339
Um dedo erguido (Figura 3.4c)	741,92	1,348
V de vitória (Figura 3.4d)	760,9	1,314
Três dedos erguidos (Figura 3.4e)	752,94	1,328
Quatro dedos erguidos (Figura 3.4f)	753,65	1,327

Fonte: Autoria própria

Tabela 4.5: Throughputs calculados para cada gesto, especificamente para a arquitetura Raspberry Pi 5 e Cyclone V (DE1-SoC)

Raspberry Pi 5 Integrada ao DE1-SoC (Cyclone V)		
Gesto FPGA	$T_{Total}(ms)$	**Throughput (gestos/s)**
Palma da mão aberta (Figura 3.4a)	121,41	8,236
Punho fechado (Figura 3.4b)	121,48	8,232
Um dedo erguido (Figura 3.4c)	121,64	8,221
V de vitória (Figura 3.4d)	121,59	8,225
Três dedos erguidos (Figura 3.4e)	121,67	8,219
Quatro dedos erguidos (Figura 3.4f)	121,95	8,200

Fonte: Autoria própria

Tabela 4.6: Throughputs calculados para cada gesto, especificamente para a arquitetura DE1-SoC

Cortex-A9 no DE1-SoC (Cyclone V)		
Gesto FPGA	$T_{Total}(ms)$	**Throughput (gestos/s)**
Palma da mão aberta (Figura 3.4a)	2.698,21	0,371
Punho fechado (Figura 3.4b)	2.698,84	0,370
Um dedo erguido (Figura 3.4c)	2.698,02	0,371
V de vitória (Figura 3.4d)	2.698,16	0,371
Três dedos erguidos (Figura 3.4e)	2.698,66	0,371
Quatro dedos erguidos (Figura 3.4f)	2.698,09	0,371

Fonte: Autoria própria

Com base nos cálculos de gestos processados por segundo (presentes nas tabelas 4.4, 4.5 e 4.6, e considerando, para fins argumentativos, apenas o menor *throughput* de gestos para cada arquitetura, pode-se observar, principalmente, que:

1. A arquitetura provida do ARM Cortex-A9 no DE1-SoC apresenta a menor taxa de processamento ($0,370$ gestos/s), com uma execução mais demorada devido ao tempo de envio dos canais de imagem à FPGA contida no *DE1-SoC*, feito por meio do arquivo que executa em espaço de usuário no Linux Yocto.

2. Já a arquitetura que utiliza somente a Raspberry Pi 5 (sem integração externa) apresenta a segunda menor taxa de gestos processados por segundo ($1,218$ gesto/s), em seu pior caso. Os resultados acumulados refletem em limitações do processamento individual performado na Raspberry Pi 5 no espaço de usuário.

3. Contudo, a arquitetura que dispõe da Raspberry Pi 5 integrada ao DE1-SoC (Cyclone V) melhora significativamente a taxa de processamento, devido à colaboração

entre a Raspberry Pi 5 e o DE1-SoC, evidenciando o impacto positivo da comunicação entre os dispositivos para acelerar a execução dos gestos. Esse fato unido ao uso da biblioteca SPIDEV, em espaço de kernel do sistema, colaboram para o melhor desempenho da arquitetura.

Os resultados indicados acima também mostram, indiretamente, que em aplicações destas arquiteturas em um sistema de tempo real, cuja *deadline* necessária para execução das tarefas seja estipulada (para fins comparativos) em um segundo, apenas as arquiteturas propostas em 3.4.2 e 3.4.3 estariam aptas para esta aplicação. Logo, a arquitetura que se mostrou de fato mais eficiente, com base nas métricas avaliadas, foi a que utilizou o processamento colaborativo entre Raspberry Pi 5 e Cyclone V (DE1-SoC).

4.2.2 Impacto da comunicação *SPI* entre dispositivos

Além disso, é importante ressaltar que apesar de, em teoria, haver uma sobrecarga adicional devido à comunicação entre os dispositivos via SPI, os resultados mostram que a arquitetura colaborativa descrita em 3.4.3 apresentou um desempenho superior em comparação com a arquitetura com a Raspberry Pi 5 independente, conforme descrito em 3.4.2. O tempo total de execução da *pipeline* na arquitetura colaborativa, com relação a Raspberry Pi 5 individual, foi significativamente menor, com uma **redução percentual de aproximadamente 89,1%** no tempo necessário para o processamento e inferência de gestos.

Por outro lado, o tempo necessário para o envio dos canais de cor da imagem na arquitetura totalmente integrada (utilizando Cortex-A9 no Cyclone V SoC) é cerca de, no pior dos casos, **29,8 vezes maior** que o observado na arquitetura colaborativa onde a Raspberry Pi 5 e o Cyclone V são combinados. Esse resultado destaca a limitação do processamento local no SoC, especialmente no que se refere ao envio de dados.

A eficiência do *SPI*, entretanto, não é apenas determinada pela escolha da arquitetura, mas também pela implementação específica do protocolo tanto na FPGA quanto no controlador (seja o Cortex-A9 ou a Raspberry Pi 5), além das configurações aplicadas ao SPI, como a frequência do clock. Configurações inadequadas, especialmente relacionadas ao clock, podem impactar diretamente a velocidade de transferência de dados e, consequentemente, o desempenho da comunicação.

As operações de envio e recebimento de pacotes de imagem, bem como as inferências realizadas via *HPS*, utilizam o conceito de *Memory Mapped I/O*. Os resultados apresentados na Tabela 4.3 reforçam que o principal gargalo da arquitetura integrada está nas operações de entrada e saída, caracterizando o sistema como *IO-Bound*.

4.3 Desafios e limitações encontradas

Nesta seção, serão discutidos os principais desafios enfrentados durante o desenvolvimento do trabalho e as limitações das arquiteturas analisadas.

4.3.1 Desafio no desenvolvimento do SPI

A configuração da comunicação via SPI representou um dos principais desafios no desenvolvimento das arquiteturas colaborativas analisadas. No caso da arquitetura baseada no Cortex-A9 (DE1-SOC), os gargalos críticos nas operações de entrada e saída comprometeram significativamente o desempenho. A ausência de otimizações no SPI, aliada às preempções do Linux Yocto, resultou em uma arquitetura incapaz de atender às demandas de tempo real. Esses problemas destacaram a necessidade de melhorias tanto na implementação do protocolo de transferência quanto no gerenciamento do sistema operacional para viabilizar sua aplicação em cenários de maior exigência.

Por outro lado, o desempenho atrelado à arquitetura colaborativa envolvendo a Raspberry Pi 5 e o FPGA Cyclone V, foi viabilizado por meio de ajustes específicos no SPI, como a definição de frequências de clock e configurações adequadas aos componentes do FPGA. Esses ajustes, realizados apenas nessa arquitetura, possibilitaram uma comunicação eficiente e estável entre os dispositivos, permitindo que a arquitetura entregasse um desempenho funcional e atendesse aos requisitos de tempo real definidos anteriormente. Neste caso, os requisitos foram arbitrariamente estabelecidos como um tempo máximo de resposta de um segundo, servindo como parâmetro para avaliar a viabilidade da arquitetura em aplicações que demandam respostas rápidas e precisas.

Em resumo, o desempenho superior da arquitetura descrita na seção 3.4.3 deve-se a otimizações específicas no SPI, como ajustes de clock e configurações de hardware, que permitiram uma comunicação eficiente e atenderam aos requisitos de tempo real. Em contraste, a arquitetura com Cortex-A9 e Cyclone V (DE1-SoC) foi prejudicada pela falta dessas otimizações e pelas limitações do sistema operacional Linux Yocto, que afetaram negativamente o desempenho e a capacidade de atender a demandas de tempo real.

4.3.2 Limitações encontradas

A arquitetura baseada no Cortex-A9 e no *FPGA Cyclone V* demonstrou limitações, principalmente, em cenários com alta demanda de transferência de dados. As operações de entrada e saída alocaram cerca de 98% do tempo total de execução, configurando o sistema como *IO-Bound*.

O protocolo SPI foi um fator limitante para a arquitetura colaborativa, apesar do desempenho superior alcançado por meio de diversas otimizações. A Raspberry Pi 5, utilizando o *spidev* e com hardware dedicado para o controle das I/Os, obteve um

desempenho mais eficiente, pois a biblioteca *spidev* opera utiliza recursos que operam ao nível de kernel. Em contraste, na arquitetura com o HPS do DE1-SoC, essas otimizações não foram implementadas, resultando em um desempenho inferior. Na FPGA, foram implementados dois tipos de *SPI*: um mais lento e estável, e outro mais rápido, mas com problemas no envio de dados. Esses fatores evidenciam que, apesar do *throughput* superior obtido pela combinação entre a Raspberry Pi 5 e o Cyclone V, a dependência do *SPI* pode trazer outros desafios em cenários que exigem alta escalabilidade ou baixa latência.

Por fim, ao analisar o desempenho da Raspberry Pi 5 como uma unidade de processamento independente, foi observado que ela possui limitações consideráveis para aplicações que exigem alta taxa de processamento em tempo real. O *throughput* foi insuficiente para atender a supostas *deadlines* abaixo de um segundo, evidenciando as restrições do processamento independente.

Capítulo 5

Conclusão

Este trabalho teve como objetivo comparar e validar diferentes arquiteturas de processamento para sistemas embarcados com requisitos de tempo real, utilizando o reconhecimento de gestos como estudo de caso.

Portanto, através da avaliação de três arquiteturas distintas, os resultados mostraram que a arquitetura colaborativa entre a Raspberry Pi 5 e o FPGA Cyclone V foi a mais eficaz, apresentando uma redução de $89,1\%$ no tempo total de execução em relação a Raspberry Pi 5 isolada, evidenciando a vantagem do processamento colaborativo.

Em contrapartida, a arquitetura baseada no Cortex-A9 teve limitações em cenários com alta taxa de transferência de dados, devido ao gargalo nas operações de entrada e saída e as preempções do sistema operacional Yocto. Em comparação com a arquitetura Raspberry 5 isolada, o tempo de execução da arquitetura baseada no Cortex-A9 foi cerca de 29,8 vezes maior. A Raspberry Pi 5 isolada também apresentou limitações em cenários com *deadlines* estritos, demonstrando que o processamento local, para a *pipeline* sugerida, não é suficiente para atender a maiores exigências.

Além disso, os objetivos estabelecidos no capítulo 1 foram alcançados, por meio da avaliação de cada arquitetura e da comparação entre as mesmas. O capítulo 4 possibilitou a análise das vantagens e limitações de cada abordagem. Embora o reconhecimento de gestos não tenha sido o foco principal deste trabalho, ele serviu como um estudo de caso relevante, permitindo a validação do desempenho das arquiteturas em cenários de processamento em tempo real.

Como sugestões para trabalhos futuros, é recomendada a exploração de protocolos de comunicação de maior largura de banda, além de otimizações no gerenciamento de I/O e melhorias no sistema operacional para mitigar os atrasos causados por preempções. Essas abordagens podem melhorar significativamente o desempenho das arquiteturas analisadas. Além disso, investigações adicionais sobre alternativas de hardware e novos modelos de arquitetura de sistemas embarcados são importantes para ampliar a aplicabilidade das soluções propostas, permitindo uma maior eficiência e robustez em cenários com requisitos de tempos reais mais exigentes.

Referências

Terasic Inc. (2019a). *DE1-SoC User Manual*. Terasic Inc. Versão acessada online.

Terasic Inc. (2019b). *Terasic- My First HPS*. Terasic Inc. Versão acessada online.

Al-Shamayleh, A., Ahmad, R., Jomhari, N., and Abushariah, M. (2020). Automatic arabic sign language recognition: A review, taxonomy, open challenges, research roadmap and future directions. *Malaysian Journal of Computer Science*, 33:306–343.

Arêdes, B. A. R. (2009). Técnicas de wavelet thresholding aplicadas no processo de denoising de imagens digitais. Dissertação de mestrado, Pontifícia Universidade Católica de Minas Gerais, Pós-Graduação em Engenharia Elétrica, Belo Horizonte.

Asano, S., Maruyama, T., and Yamaguchi, Y. (2009). Performance comparison of fpga, gpu and cpu in image processing. *FPL 09: 19th International Conference on Field Programmable Logic and Applications*, pages 126–131.

Bailey, D. G. (2023). *Design for embedded image processing on FPGAs*. John Wiley Sons, 2 edition.

Bouhoun, S., Sadoun, R., and Adnane, M. (2020). Opencl implementation of a slam system on an soc-fpga. *Journal of Systems Architecture*, 111:101825.

Carmonal, C. A. N., Sánchez-Chero, M.-J., Ortiz, E. O., Julca, J. C. S., and Ipanaqué, C. L. R. (2021). Evaluación del rendimiento de las arquitecturas de hardware hps y hps+fpga para un sistema de procesamiento de imágenes. *Revista de la Universidad del Zulia*, 12:358–373.

Embedded, C. (2017). Architecture of 8051 microcontroller. Accessed: 2024-11-29.

Gomez-Gil, P. (2009). Shape-based hand recognition approach using the morphological pattern spectrum. *Journal of Electronic Imaging*, 18:013012.

Gonzalez, Rafael C e Woods, R. E. (2017). *Digital Image Processing*. Pearson, Upper Saddle River, NJ, 4 edition.

Gupta, A., Sehrawat, V. K., and Khosla, M. (2012). Fpga based real time human hand gesture recognition system. *Procedia Technology*, 6:98–107.

Harris, L. (2015a). *Statistics for Business and Economics*. Pearson. Section 5: Testing for Significance.

Harris, L. (2015b). *Statistics for Business and Economics*. Pearson. Section 4: Kruskal-Wallis Test.

Hu, H., Zhuang, J., Zhu, E., Gao, J., and Gao, B. (2024). A real-time cardiac output monitoring system based on photoplethysmography. *Journal of Physics: Conference Series*, 2850:012006.

Intel Corporation (2013). *Cyclone V Device Handbook*. Intel FPGA (Altera). Disponível no diretório `datasheet/fpga` do conjunto DE1-SoC_v.5.1.3_HWrevF.revG_SystemCD.

Jeon, S., Park, S., Bae, J., and Lim, S. (2024). Applying multistep classification techniques with pre-classification to recognize static and dynamic hand gestures using a soft sensor-embedded glove. *IEEE Sensors Journal*.

Kamal, R. (2008). *Embedded Systems: Architecture, programming and design*. McGraw Hill Education.

Luo, W. (2022). Research on gesture recognition based on yolov5. *2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering, ICBAIE 2022*, pages 447–450.

M.E, M. H. N. (2020). Introduction to 8051 microcontroller, slide 8. Accessed: 2024-11-29.

Melo, G. (2024). Gestos estáticos analisados. Imagem não publicada, contribuição pessoal. Usada com permissão do autor.

Montgomery, D. C. (2001). *Design and Analysis of Experiments*. Wiley. Capítulo 1: Introdução.

Nascimento, P. (2024a). Códigos em linguagem c para comunicação e processamento das imagens. Disponível em: `https://github.com/paodealho404/tcc/tree/main/hps`. Acesso em: 2024-11-15.

Nascimento, P. (2024b). Códigos em linguagem python para processamento das imagens na fpga e raspberry pi 5. Disponível em: `https://github.com/paodealho404/tcc/tree/main/raspberry`. Acesso em: 2024-11-15.

Nascimento, P. (2024c). Projeto do quartus ii em verilog com todos os componentes necessários. Disponível em: `https://github.com/paodealho404/tcc/tree/main/fpga`. Acesso em: 2024-11-15.

Paratane, D. B., Patil, A., and Chaudhari, J. P. (2016). Advanced level modelling and simulation of can controller for its implementation in fpga (soc) with synthesis and timing results for integrated can node. *International Journal of Current Research and Academic Review*, 4:76–88.

PiEmbSysTech (2024). Spi communication protocol: A comprehensive guide to serial peripheral interface. Acesso em: 29 nov. 2024.

Rehman, A., Hussein, S., and Sultani, W. (2024). Joint stream: Malignant region learning for breast cancer diagnosis. *Biomedical Signal Processing and Control*, 99.

SparkFun (2024). Serial peripheral interface (spi). Acesso em: 29 nov. 2024.

Staszewski, W., Jabłoński, A., and Dziedziech, K. (2018). A survey of communication protocols in modern embedded condition monitoring systems. *Diagnostyka*, Vol. 19, No. 2:53–62.

Toda Matéria (2024). Espectro visível da luz. Disponível em: `https://static.todamateria.com.br/upload/es/pe/espectro_visivel.jpg`. Acesso em: 2 dez. 2024.

Torres, W., Santos, L., Melo, G., Oliveira, A., Nascimento, P., Carvalho, G., Neves, T., Martins, A., and Ícaro Araújo (2024). A framework for real-time gestural recognition and augmented reality for industrial applications. *Sensors 2024, Vol. 24, Page 2407*, 24:2407.

Yun, H. and Park, D. (2024). Low-power lane detection unit with sliding-based parallel segment detection accelerator for fpga. *IEEE Access*, 12:4339–4353.

Zelazko, A. (2024). Rgb color model. Acesso em: 27/11/2024.