Iuri Silva Castro

Uma Abordagem para Particionamento Hardware/Software Baseada em Reconfiguração Parcial e Dinâmica

Belo Horizonte
2019

Iuri Silva Castro

Uma Abordagem para Particionamento Hardware/Software Baseada em Reconfiguração Parcial e Dinâmica

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica.

Universidade Federal de Minas Gerais
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Orientador: Prof. Dr. Alair Dias Junior
Coorientador: Prof. Dr. Janier Arias Garcia

Belo Horizonte
2019

Castro, Iuri Silva.
C355a Uma abordagem para particionamento hardware/software baseada em reconfiguração parcial e dinâmica [recurso eletrônico] / Iuri Silva Castro. - 2019.
1 recurso online (67 f. : il., color.) : pdf.

Orientador: Alair Dias Junior.
Coorientador: Janier Arias-Garcia.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f.65-67.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Sistemas embutidos de computador - Teses. I. Dias Júnior, Alair. II. Arias-García, Janier. III. Universidade Federal de Minas Gerais. Escola de Engenharia. IV. Título.

CDU: 621.3(043)

Ficha catalográfica: Biblioteca Profº Mário Werneck, Escola de Engenharia da UFMG

"Uma Abordagem para Particionamento Hardware/software Baseada em Reconfiguração Parcial e Dinâmica"

Iuri Silva Castro

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 12 de dezembro de 2019.

Por:

Prof. Dr. Alair Dias Junior
Engenharia (TCS Industrial)

Prof. Dr. Janier Arias García
DELT (UFMG)

Prof. Dr. Diógenes Cecílio da Silva Júnior
DEE (UFMG)

Prof. Dr. Luiz Carlos Bambirra Torres
DECSI (UFOP)

Dedico a todos que me apoiaram durante toda a trajetória desse trabalho.

Agradecimentos

Agradeço aos meus pais, Jaqueline e Urciano, e minha irmã Gabriela, pelos exemplos, por me incentivarem a seguir meus sonhos, e pelo amor e companhia.

Agradeço aos meus amigos e familiares por estarem sempre presentes em minha vida.

Agradeço aos meus orientadores, Alair Dias Junior e Janier Arias Garcia, pelas orientações, pelos conselhos, pelos puxões de orelha e por acreditarem na minha capacidade.

Agradeço a empresa Invent Vision, por apoiarem e incentivarem a conclusão desse trabalho.

“Logic will get you from A to B.
Imagination will take you everywhere.”
Albert Einstein

Resumo

A mudança no paradigma computacional atual para um modelo de estrutura virtual descentralizada, conhecido como computação em nuvem, e a utilização da Internet das Coisas (IoT), está permitindo inúmeras inovações em aplicações industriais, e possibilitando a redução de custos, redução no *time-to-market* (TTM) e aumento de flexibilidade dos sistemas. No entanto, um efeito colateral desse novo paradigma é o aumento da latência, sendo ineficaz para sistemas que requerem baixa latência entre a aquisição de dados e a atuação no processo. Para esses sistemas sensíveis à latência, a computação na borda, ou *edge-computing*, se apresenta como uma solução ideal, pois parte da computação é realizada nos dispositivos de fronteira (*edge devices*). A introdução da computação na borda exige um aumento no poder computacional dos sistemas embarcados, e assim, obriga os desenvolvedores a buscarem por novas tecnologias e novas arquiteturas para atender tal necessidade. A utilização de plataformas de computação heterogêneas e um design colaborativo de hardware/software é uma solução promissora para atender tais necessidades, sendo necessário considerar, em nível de projeto, o particionamento hardware/software da aplicação. O particionamento hardware/software quando feito de forma não estruturada, causa o acoplamento das camadas de hardware e software da aplicação e leva, entre outros, a um aumento na complexidade do desenvolvimento da aplicação. Neste trabalho é apresentado uma abordagem para particionamento hardware/software utilizando reconfiguração parcial e dinâmica, onde um método de particionamento estruturado é proposto. O método proposto mostra-se capaz de suportar a utilização de reconfiguração parcial e dinâmica e de desacoplar as camadas de hardware e software da aplicação, sendo ele aplicado em um estudo de caso de um classificador de padrões, mostrando-se capaz de suportar aplicações reais da indústria.

Palavras-chave: Sistemas Embarcados, Computação na Borda, Particionamento Hardware/Software, Co-Design Hardware/Software, Reconfiguração Parcial e Dinâmica, Plataformas de Computação Heterogênea.

Abstract

The shift in the current computing paradigm to a decentralized virtual structural model, known as cloud computing, and the use of Internet of Things (IoT), is enabling numerous innovations in industrial applications, resulting in cost reduction, better time-to-market (TTM) and increased flexibility of the systems. However, a known side effect of this new paradigm is increased latency, being ineffective for systems that require low latency between acquiring data and process execution. For these latency-sensitive systems, edge-computing is an ideal solution because part of the computing is performed on edge devices. The introduction of edge-computing requires an increase in computing power of embedded systems, forcing developers to search for new technologies and new architectures to fulfill these needs. The utilization of heterogeneous computing platforms associated with a hardware/software collaborative design is a promising solution to meet such requirements, as long as we observe, at the design level, a balanced hardware/software partitioning of the application. When approached in a non-structured manner, this partitioning results in the coupling between the application's hardware and software layers and increases the application complexity, among other problems. This work presents an approach for the partitioning between hardware and software which leverages the dynamic and partial reconfiguration (DPR) features of the device. The proposed structured method was applied in a case study of a pattern classification algorithm showing that it is capable of supporting DPR and reducing the coupling between the application's hardware and software layers in real-world applications.

Keywords: Embedded Systems, Edge-Computing, Hardware/Software Partitioning, Hardware/Software Co-Design, Dynamic Partial Reconfiguration, Heterogeneous Computing Platforms.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AMBA	*Advanced Microcontroller Bus Architecture*
API	*Application Programming Interface*
ASIC	*Application-Specific Integrated Circuit*
AXI	*Advanced eXtensible Interface*
CLAS	*Classificador por Aresta de Suporte*
CPU	*Central Processing Unit*
DPR	*Dynamic Partial Reconfiguration*
DSP	*Digital Signal Processor*
FF	*Flip-Flop*
FPGA	*Field-Programmable Gate Array*
GPU	*Graphics Processing Unit*
HDL	*Hardware Description Language*
IoT	*Internet of Things*
IP	*Intellectual Property*
kNN	*k-Nearest Neighbors*
LUT	*Look-Up Table*
LVTTL	*Low-Voltage Transistor-Transistor Logic*
MR	Módulo Reconfigurável
RAM	*Random-Access Memory*
RTL	*Register-Transfer Level*
SaaS	*Software as a Service*
SO	Sistema Operacional
SoC	*System-on-Chip*
SSH	*Security SHell*

SVM	*Support Vector Machine*
TTM	*Time-to-Market*

Sumário

1 Introdução

Nos últimos anos, o paradigma computacional preponderante para sistemas organizados em rede tem mudado para uma estrutura virtual descentralizada espalhada pela rede em detrimento do modelo de estrutura física centralizada anteriormente largamente adotado. Essa estrutura virtual descentralizada é conhecida como computação em nuvem, ou *Cloud-Computing*, e proporciona fácil escalabilidade, alocação de recursos em demanda, custo reduzido de gestão e fácil provisionamento de aplicações e serviços (MOURADIAN et al., 2018).

Com o advento da Internet das Coisas, *Internet of Things* (IoT), a cada dia mais e mais dispositivos diversos são criados e conectados à rede, possibilitando, entre outros, o acesso remoto à dados de sensores, atuação em processos e interações com os usuários. Juntos, a computação em nuvem e a IoT tornam-se aliados fortes para atender as necessidades da Industria 4.0, com a descentralização da infraestrutura, sensores e equipamentos interligados proporcionando um modelo de produção inteligente e adaptativo, rápida implementação de novas funcionalidades seguindo o modelo de *Software as a Service* (SaaS) e especialização dos serviços.

Para aplicações com restrições impostas em relação à latência, à privacidade dos dados envolvidos na aplicação, e ao volume de dados produzido, a utilização de computação em nuvem pode não ser adequada (SHI et al., 2016). Exemplos de aplicações como em *smart homes*, onde dispositivos estão constantemente coletando dados das residências e de seus ambientes e, consequentemente, da vida de seus moradores, a privacidade e a segurança devem ser levadas em consideração, enquanto que em aplicações industriais, onde sistemas coletam dados de processos e fazem o seu controle em tempo real, a baixa latência entre a coleta dos dados e a atuação é uma das restrições da aplicação. Outro exemplo de aplicação é a de sistemas que geram um grande volume de dados para serem analisados, como em análises preditivas para a detecção de falhas e câmeras de monitoramento, onde o tráfego desses dados para serem analisados na nuvem aumenta os gastos em banda de dados. Nesse contexto, a computação na borda, ou *Edge-Computing*, surgiu com o intuito de que parte ou toda a computação seja feita nos dispositivos de fronteira, desta forma aliviando o consumo de banda, mantendo local dados sigilosos e reduzindo a latência (SHI et al., 2016).

Com sua utilização, a computação na borda introduz a necessidade de aumentar o poder computacional dos dispositivos de borda, principalmente para aplicações que envolvam análises de grandes e complexos volumes de dados, como aplicações que envolvam técnicas de inteligência artificial (SODHRO; PIRBHULAL; ALBUQUERQUE, 2019).

Soluções comuns disponíveis no mercado, como microprocessadores de propósito geral, devido a suas características generalistas, muitas das vezes não são capazes de entregar o poder computacional necessário e ainda manter os requisitos de tamanho, consumo e preço dentro dos limites da aplicação.

O projeto e utilização de um ASIC (*Application-Specific Integrated Circuit*), onde um

circuito projetado especificamente para atender a aplicação é fabricado em uma pastilha de silício, pode ser capaz de atender os requisitos computacionais e os requisitos físicos de tamanho e consumo da aplicação, porém o seu custo de projeto elevado e sua inflexibilidade em mudanças de requisitos da aplicação acabam desincentivando o seu uso, principalmente para baixos volumes de produção.

Uma opção ao projeto de ASIC é a utilização de arquiteturas reconfiguráveis, como FPGAs (*Field-Programmable Gate Array*). Elas permitem que circuitos específicos à aplicação sejam projetados e implementados em sua malha lógica, podendo ela ser reconfigurada, até mesmo em tempo de execução, adequando às novas funcionalidades, obtendo alto desempenho e baixo consumo de ordem de grandeza comparável a aplicações ASIC, sem sacrificar a flexibilidade.

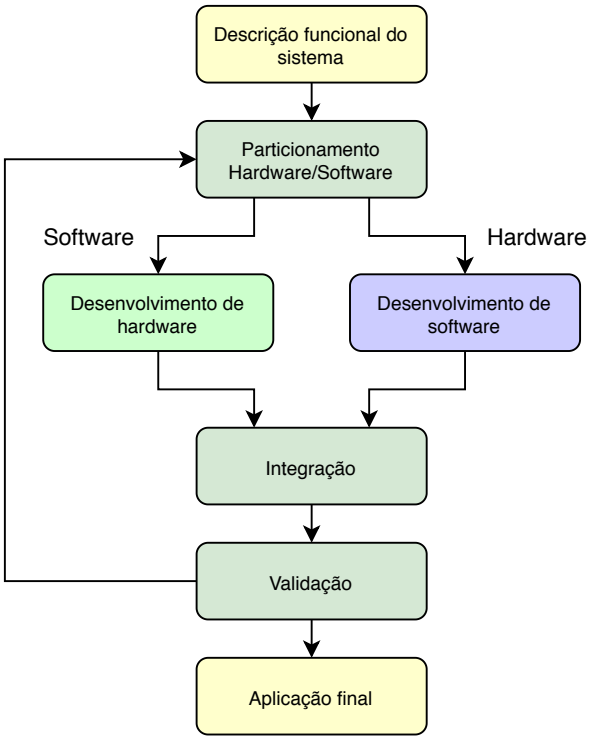
Hoje em dia encontra-se no mercado dispositivos compostos por um ou mais microprocessadores conectados a uma malha lógica de um FPGA, implementados na mesma pastilha de silício. Esses dispositivos são exemplos de plataformas de computação heterogêneas e são conhecidos pelo nome de SoC (*System-on-Chip*) FPGA.

A natureza reconfigurável dos FPGAs permite que aplicações utilizando SoC FPGA possam ser abordadas de inúmeras formas diferentes, tendo os projetistas a liberdade para escolher as funcionalidades a serem implementadas no hardware reconfigurável e no software executado pelo(s) microprocessador(es), onde as funcionalidades implementadas em hardware possuem, geralmente, um desempenho melhor que implementações em software, porém com custos de implementação, manutenção e tempo de desenvolvimento maiores. Essa divisão entre implementações em hardware e em software é conhecida como particionamento hardware/software (STITT; LYSECKY; VAHID, 2003), enquanto que as funcionalidades implementadas em hardware são conhecidas como aceleradores de hardware, ou simplesmente aceleradores, e as funcionalidades implementadas em software são conhecidas como software da aplicação.

O desenvolvimento de aplicações com particionamento pode ser feito utilizando o processo de *Concurrent Design*, ou co-design. O co-design hardware/software, como é conhecido, busca separar o desenvolvimento das implementações em hardware e software, permitindo que eles caminhem de forma concorrente, conforme pode ser visto na Figura 1, acelerando os processos de testes, verificação, integração e validação. O co-design permite que diferentes particionamentos possíveis da aplicação sejam explorados em um processo recursivo de forma a obter o melhor particionamento capaz de atender restrições, como desempenho, latência, custo e consumo, impostas pela aplicação (TEICH, 2012).

A Reconfiguração Parcial e Dinâmica, ou *Dynamic Partial Reconfiguration* (DPR), é um recurso disponibilizado em alguns dispositivos FPGA que possibilita que suas regiões possam ser reconfiguradas em tempo de execução sem interferir no funcionamento das regiões que não estão sendo reconfiguradas, podendo essas regiões continuar operantes durante todo o processo. Desta forma, a DPR permite que os circuitos implementados no FPGA se adaptem a mudanças de requerimentos das aplicações em tempo de execução de forma dinâmica, onde os circuitos

Figura 1 – Desenvolvimento típico utilizando co-design.



Fonte: Produzido pelo autor

que não vão ser reconfigurados continuam sua execução normalmente (VIPIN; FAHMY, 2018).

A utilização da DPR traz inúmeras vantagens, como (LIE; FENG-YAN, 2009; KRILL et al., 2010):

- Reutilização e multiplexação por tempo de recursos de hardware, de forma que aceleradores que não são executados simultaneamente podem utilizar a mesma região do FPGA sendo reconfigurados sob demanda, ocorrendo desta forma uma virtualização dos recursos, ou seja, a aplicação enxerga mais recursos do que existem fisicamente;
- Redução da área e consequentemente redução na potência dissipada pelo sistema, permitindo que sejam utilizadas FPGAs com malhas lógicas menores;
- Redução do tempo de reconfiguração, possibilitando ganhos em aplicações com restrições temporais;
- Ganho em escalabilidade do sistema, onde novas funcionalidades podem ser posteriormente implementadas e inseridas em sistemas em operação.

Apesar das vantagens da DPR, sua utilização ainda não é amplamente difundida em aplicações comerciais devido à complexidade envolvida nas fases de projeto e implementação, e às limitações ainda existentes das ferramentas de desenvolvimento e nas arquiteturas dos

dispositivos, sendo ainda predominantemente utilizadas em estudos acadêmicos. Os autores Vipin e Fahmy (2018) enumeraram alguns aspectos que ainda faltam para que a DPR seja mais difundida em aplicações comerciais, destacando-se:

1. Suporte para a realocação dos arquivos de configuração (*bitstreams*) em tempo de execução, não sendo necessário gerar diferentes versões da mesma configuração para cada alocação possível;
2. Operação de reconfiguração transparente para a aplicação, de forma que a aplicação continue executando normalmente durante a reconfiguração e não se preocupe com os detalhes de implementação da reconfiguração;
3. Ferramentas de desenvolvimento em alto-nível capaz de automatizar o processo de mapeamento e alocação dos recursos do dispositivo para os aceleradores, sem que seja necessário que o projetista conheça a arquitetura do dispositivo em baixo-nível.

1.1 Formulação do Problema

A utilização do particionamento hardware/software introduz novos desafios para o desenvolvimento das aplicações, como a necessidade de definir as formas de comunicação entre os aceleradores e o software da aplicação, avaliando as interfaces de hardware, como os barramentos, e as interfaces de software, como os protocolos, necessárias. O particionamento torna-se ainda mais desafiador quando utiliza-se a reconfiguração parcial e dinâmica, pois a reconfiguração parcial impõe que os aceleradores possuam as mesmas interfaces de hardware para que eles sejam compatíveis com as mesmas regiões de reconfiguração, além de necessitar que as regiões de reconfiguração sejam definidas em tempo de projeto.

A possibilidade de múltiplas interações no processo de particionamento, devido ao co-design, aumenta o custo da etapa de integração no desenvolvimento e pode introduzir acoplamento entre a camada de hardware, composta pelos aceleradores, e a camada de software, composta pelo software da aplicação, provocando uma interdependência entre as funcionalidades e dificultando o desenvolvimento, a manutenção, a reutilização e a portabilidade das aplicações, sendo necessário definir interfaces de software para reduzir o tempo de integração e evitar o acoplamento das camadas.

Dessa forma, são abordados nesse trabalho os problemas relacionados ao particionamento hardware/software com reconfiguração parcial e dinâmica utilizados em um processo de co-design hardware/software.

1.2 Objetivos

O objetivo geral desse trabalho é propor um método para o particionamento hardware/software utilizando reconfiguração parcial e dinâmica que auxilie no processo de integração

evitando acoplamento entre as camadas. Assim, o método é composto por definições de padrão para as interfaces de hardware dos aceleradores, uma arquitetura de suporte para projetar as regiões reconfiguráveis e suas conexões, padronizações das interfaces de software em conjunto com uma *Application Programming Interface* (API), e uma metodologia de desenvolvimento que suporta todo o ferramental desenvolvido.

1.2.1 Objetivos Específicos

Dos objetivos específicos, pode-se citar:

A. Definir um padrão para as interfaces de hardware dos aceleradores;

B. Desenvolver uma arquitetura de suporte para organizar as regiões reconfiguráveis e suas conexões;

C. Definir um padrão para as interfaces de software dos aceleradores;

D. Desenvolver a API para fornecer funções de acesso, controle e reconfiguração dos aceleradores;

E. Desenvolver uma metodologia capaz de guiar o processo de implementação do método proposto em aplicações reais.

1.3 Metodologia

Para alcançar os objetivos, pretende-se:

1. Levantar os requisitos de comunicação dos aceleradores e padronizar suas interfaces de hardware;

2. Elaborar uma arquitetura de suporte capaz de atender os requisitos para utilizar reconfiguração parcial e dinâmica;

3. Levantar os requisitos de comandos e controle dos aceleradores e padronizar suas interfaces de software;

4. Desenvolver a API;

5. Elaborar uma metodologia para adequar aplicações no método proposto;

6. Utilizar um estudo de caso para validar o método proposto.

1.4 Contribuições

São contribuições desse trabalho:

1. Um método que permite a utilização de reconfiguração parcial e dinâmica no processo de particionamento hardware/software;
2. Uma API que possibilita a gerência transparente de aceleradores implementados em hardware a partir do software da aplicação, facilitando a etapa de integração no processo de co-design hardware/software;
3. Uma metodologia para guiar desenvolvedores na utilização do método proposto em suas aplicações;
4. Um estudo de caso para aplicações envolvendo classificadores de padrões.

1.5 Estrutura do Texto

Primeiramente, uma revisão da literatura é apresentada no capítulo 2, com conceitos abordados nesse trabalho e trabalhos relacionados. O capítulo 3 descreve o processo de desenvolvimento do método proposto, mostrando as padronizações, definição de arquitetura de suporte, API e metodologia. O capítulo 4 apresenta o estudo de caso utilizado para validar o método proposto, mostrando e discutindo os resultados obtidos. O capítulo 5 expõe as conclusões do trabalho e as possibilidades para trabalhos futuros.

2 Revisão da Literatura

Inicialmente apresenta-se alguns conceitos abordados no trabalho, e, em seguida, alguns trabalhos relacionados encontrados na literatura.

2.1 Conceitos

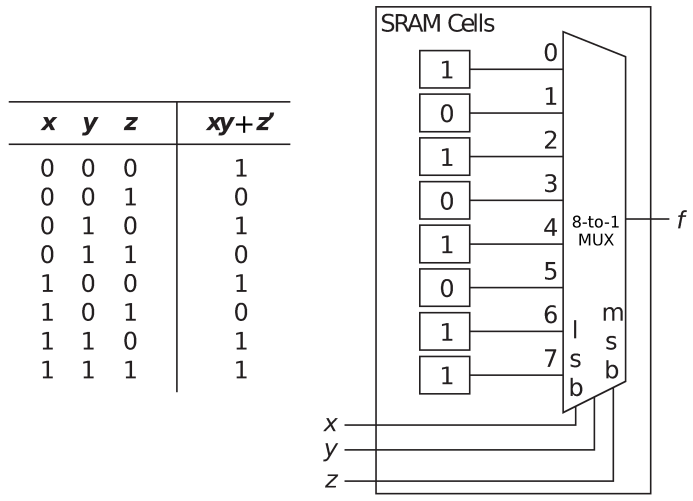
As subsessões abaixo abordam os principais conceitos apresentados nesse trabalho.

2.1.1 FPGA

Os FPGAs são dispositivos compostos por blocos lógicos programáveis, blocos lógicos com funções fixas e blocos de entrada e saída, interligados por uma rede de interconexões programável em uma forma de matriz, formando uma malha lógica.

Os blocos lógicos programáveis são capazes de implementar funções lógicas combinacionais, lógicas sequenciais, e armazenar valores, sendo compostos por LUTs (*Look-Up Table*) e FFs (*Flip-Flops*). A LUT é implementada utilizando circuitos multiplexadores e memória, sendo a memória utilizada para armazenar a tabela lógica da função implementada e os multiplexadores para fazer a seleção da entrada da tabela, vide exemplo da Figura 2. A utilização de *Flip-Flops* permite que lógicas sequenciais sejam implementadas, ou mesmo valores possam ser armazenados pelos blocos lógicos, expandindo as possibilidades de utilização dos blocos (SASS; SCHMIDT, 2010).

Figura 2 – LUT de uma função lógica combinacional de 3 entradas.



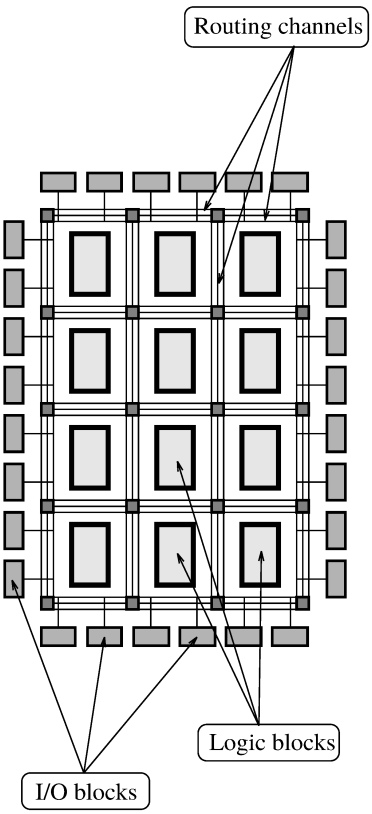
Fonte: Sass e Schmidt (2010)

Além dos blocos lógicos programáveis, encontra-se também nos dispositivos blocos lógicos com funções fixas, como multiplicadores, somadores, memórias, geradores de *clock*, entre outros. Os tipos e quantidades de cada um desses blocos variam de acordo com o fabricante e a arquitetura do dispositivo, sendo eles, normalmente, distribuídos de forma não uniforme pela malha lógica do dispositivo.

Para fazer interface com dispositivos externos, os FPGAs possuem blocos de entrada e saída, localizados nas extremidades da malha lógica, sendo tais blocos especializados em interfaces físicas, podendo ser programados para entrada e/ou saída de sinais e atender níveis diferentes de tensão e corrente de acordo com padrões existentes na industria, como, por exemplo, o LVTTL (*Low-Voltage Transistor-Transistor Logic*) (SASS; SCHMIDT, 2010).

As conexões entre os blocos lógicos que compõe o FPGA são feitas através de interconexões programáveis, possibilitando desta forma rotear sinais e agrupar diversos blocos para poder implementar a funcionalidade desejada. A Figura 3 exemplifica uma malha lógica genérica de um FPGA.

Figura 3 – Malha lógica do FPGA.



Fonte: Meyer-Baese (2013). Modificado pelo autor

Assim como na LUT, as interconexões programáveis e as funcionalidades dos blocos lógicos são configuradas a partir de elementos de memória, podendo esses elementos serem voláteis ou não-voláteis. Atualmente, os principais dispositivos comercializados utilizam memória

estática volátil, SRAM, e é necessário que sejam configurados toda vez que sua alimentação seja aplicada.

2.1.2 Fluxo de Projeto

O fluxo de projetos básico para FPGAs começa com a descrição do circuito desejado, sendo ainda predominante a utilização de linguagens de descrição de hardware, *Hardware Description Language* (HDL), como Verilog HDL e o VHDL, com o método de descrição *Register-Transfer Level* (RTL) (LAHTI et al., 2019).

O circuito descrito passa por uma ferramenta de síntese, onde é gerado uma lista contendo os blocos lógicos e suas conexões necessárias, sendo conhecida como *netlist*. A *netlist* é então utilizada em ferramentas de mapeamento tecnológico em um processo conhecido como *place-and-route*, onde os recursos utilizados na *netlist* são separados e mapeados no dispositivo e suas rotas definidas.

É permitido que o projetista interfira no processo de *place-and-route*, indicando para a ferramenta as regiões onde ela deverá implementar a *netlist* ou parte dela. Esse processo é conhecido como *floorplanning*, e é utilizado por peritos na área principalmente para ter ganhos em consumo e velocidade, e, no nosso caso, para indicar regiões que serão parcialmente reconfiguradas.

Ao final, a validação temporal do circuito (*timing*) é feita, verificando se os atrasos devidos ao posicionamento e distância das rotas não interferem nos requisitos do circuito e é gerado um vetor de bits conhecido como *bitstream* com os valores a serem escritos na memória de configuração do FPGA.

2.1.3 Reconfiguração

A reconfiguração dos dispositivos FPGAs pode ser classificada em dois tipos: a reconfiguração total e a reconfiguração parcial. A reconfiguração parcial ainda pode ocorrer de forma estática ou dinâmica.

A reconfiguração total é o processo no qual toda a configuração da malha lógica do FPGA é sobrescrita para armazenar a nova configuração, sendo o FPGA mantido inoperante até o final do processo. Devido aos dispositivos utilizarem memórias voláteis para manter a configuração da malha lógica do FPGA, o processo de reconfiguração total é necessário toda vez que ocorra a reinicialização do FPGA por interrupção de alimentação. A reconfiguração total também pode ser acionada após a inicialização do FPGA utilizando dispositivos externos, como microcontroladores, que tenham acesso físico a porta de configuração do FPGA (MAXFIELD, 2004).

A reconfiguração parcial, por outro lado, permite que partes do FPGA sejam reconfiguradas sem que seja necessário reconfigurar todo o dispositivo, podendo a reconfiguração ocorrer de forma estática, onde mesmo as partes que não estão sendo reconfiguradas são mantidas

inoperantes durante o processo, ou de forma dinâmica, onde as partes que não estão sendo reconfiguradas são mantidas operantes e não são afetadas pelo processo de reconfiguração.

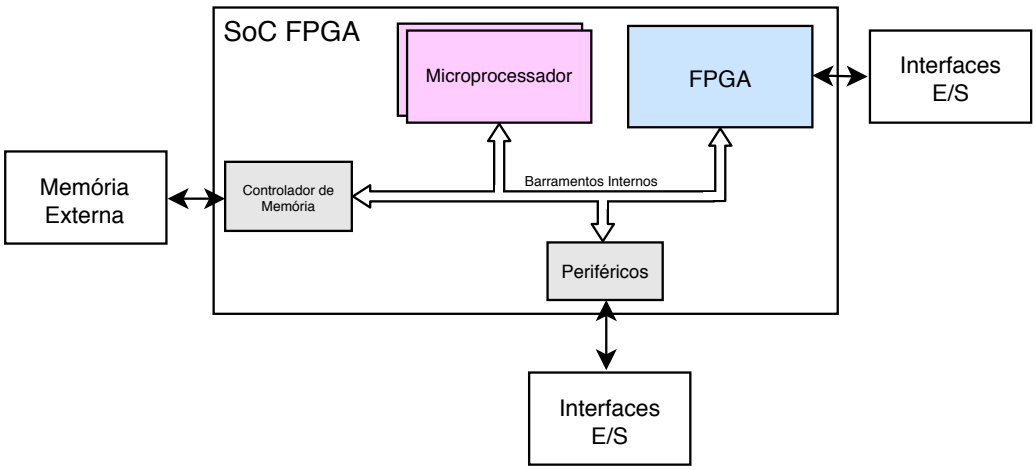
Nem todos os fabricantes disponibilizam no mercado dispositivos que suportam a reconfiguração parcial e dinâmica, sendo a Xilinx (XILINX, 2019) e a Intel FPGA (INTEL, 2019) os maiores fabricantes que atualmente disponibilizam dispositivos que possuem tal funcionalidade.

2.1.4 Computação Heterogênea

Plataformas de computação heterogênea são sistemas compostos por diferentes tipos de unidades de processamento, como CPUs (*Central Processing Unit*), GPUs (*Graphics Processing Unit*) e DSPs (*Digital Signal Processor*), e arquiteturas reconfiguráveis, como FPGAs, trabalhando em conjunto para atender as necessidades de desempenho e consumo energético de determinada aplicação, explorando as diferentes características de cada unidade.

Em aplicações de sistemas embarcados, está se tornando comum a utilização de dispositivos que implementem plataformas de computação heterogêneas compostas por múltiplos processadores conectados à uma malha lógica de um FPGA, conhecidos como SoC FPGA (AFONSO et al., 2013). Nesses dispositivos SoC FPGA, barramentos internos compartilhados de alta velocidade permitem que os microprocessadores e o FPGA se comuniquem e acessem memórias externas compartilhadas e periféricos também presentes no dispositivo. A Figura 4 mostra uma arquitetura genérica de um dispositivo SoC FPGA, detalhes da arquitetura interna variam por fabricante e família do dispositivo em questão.

Figura 4 – Exemplo de uma arquitetura genérica de um SoC FPGA contendo múltiplos microprocessadores conectados à um FPGA.



Fonte: Produzido pelo autor

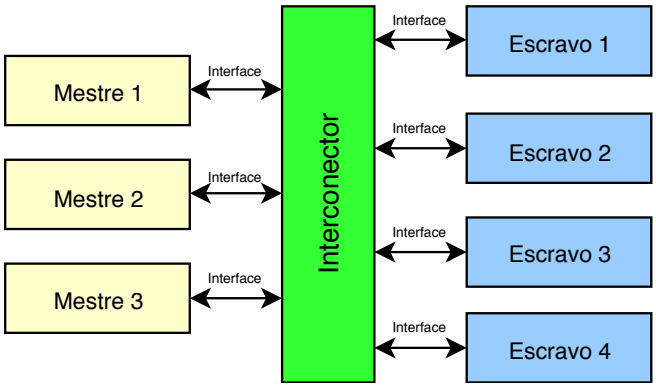
2.1.5 AMBA AXI4

A interface AMBA (*Advanced Microcontroller Bus Architecture*) implementando o protocolo AXI (*Advanced eXtensible Interface*), ambos desenvolvidos pela empresa ARM, é capaz de fornecer alto desempenho e alta banda de dados com baixa latência (ARM, 2011). O protocolo é baseado em transferências em rajadas (*burst*), com cinco canais independentes para a transferência de endereço de escrita, endereço de leitura, dados de escrita, dados de leitura e resposta de escrita, permitindo leituras e escritas simultâneas e disparar múltiplas requisições de endereços de escritas e leituras.

O protocolo permite um barramento com a largura de dados de até 1024-bits e rajadas de até 256 palavras, sendo a palavra do tamanho do barramento, sem especificar a frequência máxima de operação do barramento. O limite teórico de banda é de uma palavra de barramento por ciclo de *clock* e possibilita que uma transação seja concluída em dois ciclos de *clock*.

A interface é amplamente utilizada na interconexão de periféricos com um ou mais processadores em dispositivos e segue o padrão *mestre-escravo*, podendo o barramento conter múltiplos mestres e múltiplos escravos, possuindo um interconector para mediar as transações, como pode ser visto na Figura 5.

Figura 5 – Interconector AXI.



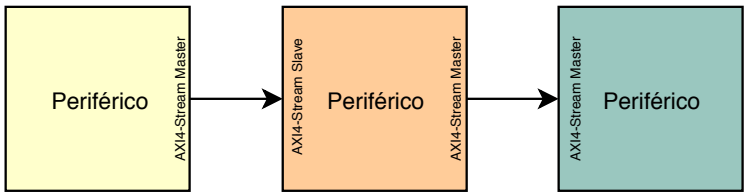
Fonte: Produzido pelo autor

Para a quarta versão da especificação, AMBA AXI4, a interface implementa três variações do protocolo:

- AXI4-Full, ou apenas AXI4, contemplando a especificação completa do protocolo, com seus acessos mapeados em memória e possibilitando transações de *burst*;
- AXI4-Lite, sendo um subconjunto mais simples da especificação, com seus acessos mapeados em memória e simples implementação em hardware, porém sem suporte à transações de *burst*;

- AXI4-Stream, sendo um protocolo para transferência de dados em cadeia, unilateral, entre periféricos (ARM, 2010), vide Figura 6.

Figura 6 – Exemplo de periféricos utilizando AXI4-Stream para transferência de dados em cadeia.

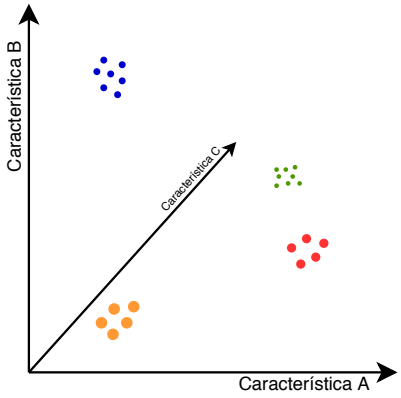


Fonte: Produzido pelo autor

2.1.6 Classificadores de Padrões

A classificação de padrões vem do conceito de discriminar em classes entidades como objetos, eventos, indivíduos, entre outros, de acordo com suas características mensuráveis, como por exemplo o formato, o tempo de duração, as dimensões, a temperatura e a cor. Utilizando-se das características, um espaço multidimensional pode ser gerado, onde os seus eixos são representados pelas características e, desta forma, as amostras das entidades podem ser representadas como pontos nesse espaço. A Figura 7 mostra um exemplo de uma entidade sendo representada por três características (A, B e C), compondo um espaço tridimensional.

Figura 7 – Exemplo de representação de entidades com três características em um espaço tridimensional.



Fonte: Produzido pelo autor

Com a representação em um espaço multidimensional, torna-se possível utilizar métricas para analisar o grau de similaridade entre elas, como, por exemplo, a distância entre as amostras.

A área de Aprendizado de Máquina, ou *Machine Learning*, busca estudar métodos e algoritmos capazes de tomar decisões a partir de conhecimentos previamente adquiridos através de um processo de aprendizagem, também conhecido como treinamento (MITCHELL, 1997). Entre os tipos de aprendizagem, destacam-se o aprendizado supervisionado e o não supervisionado, onde no aprendizado supervisionado têm-se o conhecimento a priori das possíveis saídas e são utilizados no treinamento conjuntos de dados de amostras e suas saídas esperadas, enquanto que no aprendizado não supervisionado não há o conhecimento a priori das possíveis saídas e nem das saídas esperadas para o conjunto de dados de treinamento.

Os classificadores são algoritmos que, através do aprendizado supervisionado, utilizam os dados de treinamento para induzir um modelo matemático de uma função, conhecida como função classificadora, que atuará no espaço multidimensional que representa as entidades. Após o treinamento, a função classificadora é utilizada para determinar a classe de novos dados de entrada, conhecidos como amostras de teste.

Existem vários tipos diferentes de classificadores, sendo eles caracterizados pelos modelos e métodos abordados, como modelos estatísticos e modelos lineares, assim como a quantidade de classes distintas de classificação, podendo ser do tipo binário (duas classes) ou do tipo multiclasse (três ou mais classes). Nas subseções abaixo, aborda-se os classificadores de margem larga, sendo um classificador simples e seus conceitos utilizados como base para outros classificadores, e o classificador *NN-clas* proposto por Gade (2018) que baseia-se em classificadores de margem larga.

2.1.6.1 Classificadores de Margem Larga

Em um espaço multidimensional com dimensão M, um hiperplano é definido como um subespaço de dimensão $M - 1$ contido no espaço multidimensional, sendo válido para $M \in \mathbb{Z}, M \geq 2$. Como exemplo, observa-se que em um espaço bidimensional ($M = 2$) o hiperplano é definido como uma curva, enquanto que em um espaço tridimensional ($M = 3$) ele é definido como um plano. Desta forma, hiperplanos podem ser utilizados para separar, em duas regiões, o espaço multidimensional ao qual ele pertence, podendo ser utilizado intuitivamente como um classificador binário (JAMES et al., 2013).

A Equação 2.1 define um hiperplano com n dimensões, onde β são os coeficientes do hiperplano e x as variáveis.

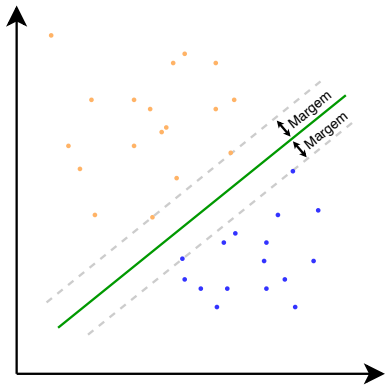
$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n = 0 \tag{2.1}$$

Os classificadores de margem larga são classificadores intuitivos com fronteiras de decisão linear e partem da ideia de utilizar hiperplanos para fazer a separação entre as classes, buscando selecionar o hiperplano que tenha a maior margem de separação entre amostras de classes distintas na região de fronteira de decisão, como pode ser visto na Figura 8. Os pontos localizados nas extremidades das margens são conhecidos como vetores de suporte, e

são utilizados para gerar o hiperplano de separação. Métricas de distância, como a distância euclidiana (Equação 2.2), são utilizadas para avaliar o grau de proximidade entre as amostras.

$$D(X,Y) = \sqrt{\sum_{n=1}^{M}(x_i - y_i)^2} \tag{2.2}$$

Figura 8 – Margens de separação entre duas classes.



Fonte: Produzido pelo autor

O critério de classificação é feito a partir do hiperplano, de acordo com a Equação 2.3, avaliando-se $f(X)$ onde $X = (x_1, x_2, ..., x_n)$ representa a amostra a ser classificada e $f(X) > 0$ e $f(X) < 0$ são as duas condições de classificação.

$$f(X) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n = 0 \tag{2.3}$$

Para o suporte de fronteiras de decisão não lineares, uma generalização dos classificadores de margem larga conhecida como SVM, ou *Support Vector Machine*, é utilizada, e busca abordar a não linearidade da fronteira de decisão inserindo termos não lineares como quadráticos e cúbicos na equação (Equação 2.1) do hiperplano (JAMES et al., 2013).

2.1.6.2 Classificador *NN-clas*

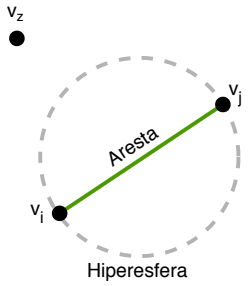
O classificador NN-clas proposto por Gade (2018) é uma extensão do classificador por arestas de suporte (CLAS) (TORRES, 2016), sendo um classificador de margem larga binário que baseia-se na distância para avaliar o grau de proximidade entre as amostras, utilizando-se da estrutura de um grafo planar conhecido como grafo de Gabriel para montar a estrutura de separação entre as classes e de um critério de decisão similar ao do classificador de vizinhos próximos (kNN).

O grafo de Gabriel é utilizado de forma que o conjunto de vértices V do grafo represente o conjunto de amostras de treinamento P, ou seja $V = P$. A condição de aresta do grafo é

definida pela Equação 2.4, onde uma aresta (v_i, v_j) pertence ao conjunto de arestas A se e somente se nenhum outro vértice v_z estiver contido na hiperesfera com diâmetro definido pela distância euclidiana das amostras representadas pelos vértices v_i e v_j, conforme pode ser visto na Figura 9.

$$(v_i, v_j) \in A \leftrightarrow D^2(v_i, v_j) \leq [D^2(v_i, v_z) + D^2(v_j, v_z)] \forall z \in V, v_i, v_j \neq v_z \tag{2.4}$$

Figura 9 – Hiperesfera formado pelos vértices $\{v_i, v_j\}$.



Fonte: Produzido pelo autor

No conjunto de arestas do grafo de Gabriel existe um subconjunto de arestas conhecidas como arestas de suporte. As arestas de suporte são definidas como as arestas no qual os vértices que a compõe pertencem à classes distintas e, desta forma, estão localizadas na margem de separação entre as duas classes, sendo os vértices das arestas de suporte análogos aos vetores de suporte dos classificadores de margem larga (TORRES, 2016).

O critério de classificação do NN-clas utiliza uma abordagem semelhante ao classificador por k-vizinhos próximos, ou *k-Nearest Neighbors* (kNN), com $k = 1$, onde as distâncias entre as amostras de teste e os vértices das arestas de suporte são calculadas e o vértice com a menor distância da amostra de teste selecionado, sendo a classe da amostra de teste rotulada com a mesma classe do vértice selecionado, ou seja, do vértice vizinho mais próximo a ela.

2.2 Trabalhos Relacionados

A utilização de plataformas de computação heterogêneas explorando o particionamento hardware/software para o alto desempenho é um tópico que vem chamando a atenção na literatura, principalmente com o crescimento em demanda por poder computacional em plataformas embarcadas.

Alguns trabalhos encontrados na literatura implementaram seus sistemas utilizando dispositivos microcontroladores e FPGAs discretos, fisicamente separados e conectados por

interfaces padrões no mercado, ou por interfaces proprietárias. Um exemplo é o trabalho dos autores Govil e Chowdhury (2014), que buscaram implementar um algoritmo de FFT (*Fast Fourier Transform*) em um sistema micro-controlado utilizando um externo FPGA conectado ao microcontrolador para absorver parte da carga do algoritmo, conseguindo, assim, uma redução de até 30% no tempo de execução comparado com a execução total feita no microcontrolador.

Os autores Ngo et al. (2013), dispostos apenas com um dispositivo FPGA, exploraram a característica reconfigurável do dispositivo e compuseram o sistema heterogêneo de sua aplicação utilizando um microprocessador implementado na própria malha lógica do FPGA, conhecido como *soft-core*. A aplicação proposta no trabalho é a de um protótipo de um equipamento de vigilância de vídeo em tempo real, utilizando algoritmos de detecção e rastreio de objetos. A opção pela arquitetura é devido a necessidade do particionamento hardware/software, onde o hardware fica responsável por executar os algoritmos de processamento dos quadros de vídeo e o software responsável por atender a interface com o usuário e fazer o controle do fluxo de dados e parâmetros do sistema. Com esse particionamento, os autores conseguiram alcançar os requisitos temporais para a análise em tempo real das imagens da câmera.

Alguns autores exploraram a reconfiguração total dos dispositivos reconfiguráveis de seus sistemas para conseguirem adaptar suas aplicações a mudanças de carga. No trabalho proposto por Xu e Schafer (2017), os autores abordaram um sistema de computação aproximada que é capaz de adaptar seus circuitos de aproximação de acordo com a mudança das características dos dados de entrada do sistema, visto que esses circuitos são otimizados para bases específicas de dados, mudanças nas características dos dados podem resultar em margens de erro não aceitáveis para a aplicação. Desta forma, o sistema proposto por eles é capaz de detectar mudanças nos dados de entrada e reconfigurar o circuito de aproximação adequado para aquele padrão de dados. Os autores justificam o tempo gasto na reconfiguração total com o longo tempo entre as mudanças nas características dos dados.

Já abordando sistemas adaptativos com requisitos de tempo real, os autores Yoon, Joung e Lee (2016), em seu sistema de detecção em tempo real de bordas em vídeos de 1080p, conseguiram adaptar, em tempo real, o filtro utilizado no quadro de vídeo de acordo com o nível de densidade de ruído detectado. Assim, com a mudança da densidade de ruídos na imagem, a aplicação consegue reconfigurar o filtro implementado no FPGA para melhor atender o nível de ruido presente. Com isso, os autores conseguiram ter ganhos na acurácia da detecção de bordas mantendo o processamento em tempo real.

Alguns trabalhos na literatura buscaram facilitar a integração e desenvolvimento de aplicações utilizando aceleradores através da virtualização do hardware. Kelm e Lumetta (2008) desenvolveram uma extensão para Linux nomeada HybridOS, possibilitando que desenvolvedores de software da aplicação possam fazer chamadas ao hardware através de uma API, tornando transparente toda a implementação do hardware e a comunicação, deixando sob responsabilidade do sistema operacional o escalonamento e compartilhamento dos aceleradores. Similarmente, Vatsolakis e Pnevmatikatos (2017) desenvolveram uma API nomeada RACOS, dando suporte à

carga/descarga dos aceleradores, assim como escalonamento e comunicação transparentes ao usuário.

3 Desenvolvimento

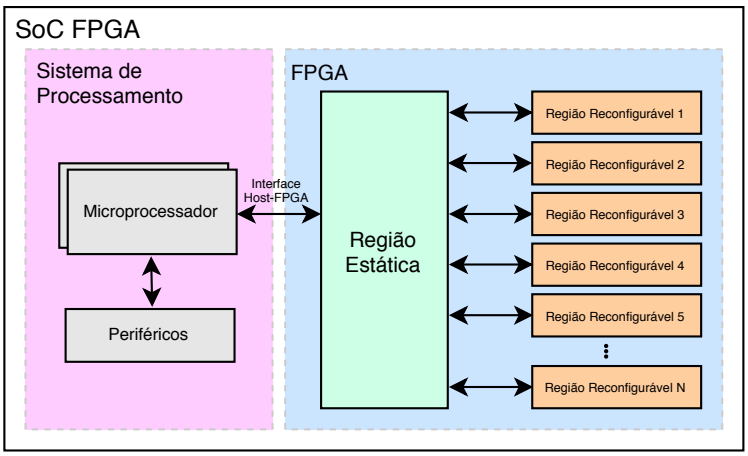
O desenvolvimento é divido em três partes, sendo a primeira parte as definições das interfaces de hardware dos aceleradores e a arquitetura de suporte, a segunda parte as definições da interface de software dos aceleradores e a implementação de uma API, e a terceira parte a definição de uma metodologia para aplicar as padronizações propostas em aplicações reais. Nomeiam-se as partes, respectivamente, como, definições de hardware dos aceleradores, definições de software dos aceleradores, e metodologia de aplicação. As seções seguintes abordam as três partes individualmente.

3.1 Definições de Hardware dos Aceleradores

A utilização de reconfiguração parcial e dinâmica requer que sejam definidas durante a fase de projeto as regiões do FPGA que poderão ser reconfiguradas em tempo de execução e as regiões fixas durante toda a execução, conhecidas respectivamente como regiões reconfiguráveis e regiões estáticas.

As regiões reconfiguráveis são as regiões onde serão mantidos os aceleradores de hardware, possibilitando a reconfiguração dos mesmos durante a execução da aplicação, enquanto que nas regiões estáticas são as regiões onde serão mantidos as interfaces de hardware com o sistema de processamento e as interfaces de hardware com os aceleradores presentes nas regiões reconfiguráveis, sendo ela responsável pela ponte de comunicação entre sistema de processamento e os aceleradores, como pode ser visto na Figura 10.

Figura 10 – Arquitetura geral do sistema.



Fonte: Produzido pelo autor

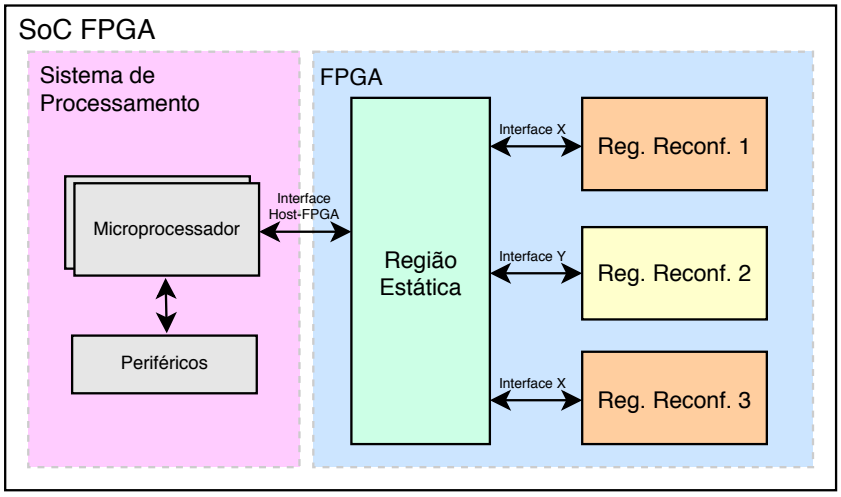
Nas subseções abaixo, faz-se as considerações relacionadas às interfaces de hardware dos aceleradores e a arquitetura suporte.

3.1.1 Interfaces de Hardware

As interfaces de hardware impactam diretamente o desempenho e os métodos de controle dos periféricos. Padrões de interfaces, como, por exemplo, o PCIe, permitem que vários periféricos diferentes possam migrar facilmente entre plataformas de hardware distintas, desde que as plataformas e os periféricos implementem o mesmo padrão. Da mesma forma, para que múltiplos aceleradores sejam compatíveis com a mesma região reconfigurável eles devem implementar a mesma interface de hardware com a região estática.

A padronização permite que seja otimizado a utilização das regiões reconfiguráveis, como pode ser visto no exemplo da Figura 11, onde a malha lógica do FPGA é dividida em três regiões reconfiguráveis onde duas utilizam a interface de hardware X e uma a interface de hardware Y, e desta forma, caso tenha três aceleradores que implementem a interface X para serem reconfigurados, apenas dois poderão ser reconfigurados ficando um aguardado o fim da execução de um dos dois para que ele possa ser reconfigurado em seu lugar, mesmo que a região que utilize a interface Y esteja disponível.

Figura 11 – Região estática, regiões reconfiguráveis e suas interfaces.



Fonte: Produzido pelo autor

Para a padronização da interface de hardware, primeiramente levanta-se a comunicação básica necessária para o funcionamento de um acelerador em hardware. Desta forma, levantou-se os seguintes itens:

- Comunicação para controle, *status* e parametrização, com requisitos de banda baixa;
- Comunicação para dados de entrada e saída dos algoritmos com requisitos de banda alta.

Assim, dadas as necessidades de comunicação dos aceleradores, e utilizando-se as diferentes implementações da interface AMBA AXI4, foram escolhidas duas interfaces padrões para os aceleradores, sendo, então, uma interface AXI4-Lite Slave (escravo) com largura de dados de 32-bits e 16 registros mapeados em memória que será utilizada pelo sistema de processamento para o controle e relatórios de *status*, e uma interface AXI4 Master (mestre) com largura de dados de 32-bits utilizada pelo próprio acelerador para fazer acesso direto à memória, podendo utilizar transações de *burst* de leitura e escrita, buscando os dados para serem processados e armazenando dados já processados.

Após definir as interfaces de hardware padrões, define-se então o módulo reconfigurável (MR) como uma região reconfigurável que implementa as interfaces de hardware padronizadas, sendo então o MR um receptáculo de aceleradores que sigam a padronização proposta.

3.1.2 Arquitetura de Suporte

A arquitetura de suporte é definida por um conjunto finito de módulos reconfiguráveis (MRs) e interconectores AXI utilizados para conectá-los ao sistema de processamento.

Os MRs compõem as regiões reconfiguráveis, com o número máximo de MRs dependendo da quantidade de recursos disponíveis na malha lógica do FPGA e dos tipos e quantidade de recursos utilizados pelos aceleradores que ele dará suporte. Não há a necessidade dos MRs serem homogêneos, ou seja, todos possuírem o mesmo tamanho e a mesma quantidade e tipo de recursos. É associado a cada MR um endereço de memória base de 32-bits utilizado pela interface AXI4-Lite Slave para mapear seus registradores em memória.

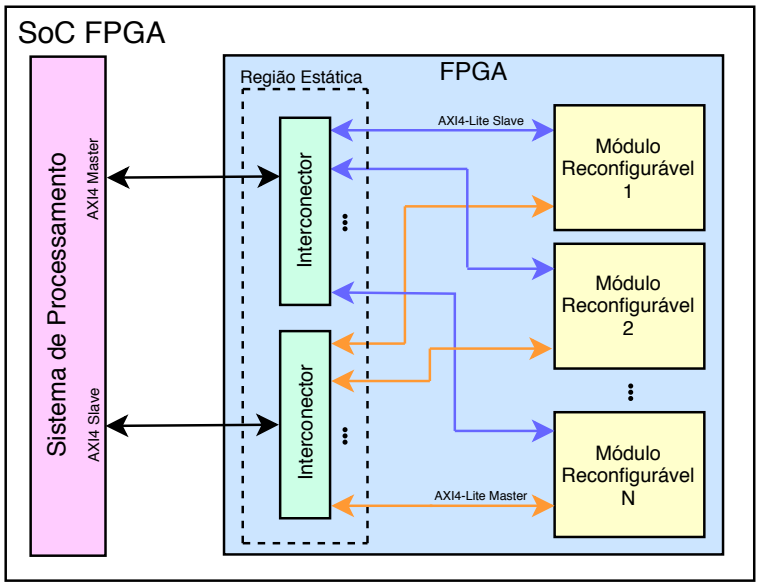
Os interconectores AXI compõem a região estática e não são opcionais à arquitetura, devendo ser obrigatoriamente implementados, sendo responsáveis por mediar as interfaces dos aceleradores com o sistema de processamento.

A Figura 12 exemplifica uma arquitetura de suporte de um sistema com um número N de módulos reconfiguráveis e interconectores AXI implementados na região estática da malha do FPGA, com os interconectores conectando-se ao sistema de processamento diretamente através da interface AXI. Para casos que a arquitetura do dispositivo SoC FPGA utilize outra interface entre o sistema de processamento e a malha do FPGA, é necessário incluir na região estática conversores de barramento entre a interface AXI e o padrão utilizado pelo dispositivo.

3.2 Definições de Software dos Aceleradores

Com a interface de hardware definida, parte-se para as definições da interface de software dos aceleradores, definindo o protocolo de comunicação utilizado entre os aceleradores e o sistema de processamento e, em seguida, criando-se a API para disponibilizar as funções de acesso e controle dos aceleradores.

Figura 12 – Exemplo de arquitetura de suporte com N módulos reconfiguráveis.



Fonte: Produzido pelo autor

3.2.1 Interfaces de Software

Uma das interfaces de hardware anteriormente padronizadas para ser utilizada no controle e acesso dos aceleradores, a interface AXI4-Lite Slave, utiliza um modelo de comunicação baseado em registros mapeados em memória. Desta forma, a interface disponibiliza um conjunto de registros que podem ser acessados pelo tanto sistema de processamento e como pelo acelerador, e assim é possível definir uma interface de software baseado na leitura e escrita dos registros, onde os parâmetros e comandos são escritos pelo sistema de processamento nos registros e interpretados pelo aceleradores, e os resultados escritos pelos aceleradores e interpretados pelo sistema de processamento.

Para definir a interface de software baseada em registros, primeiramente levanta-se os os campos e comandos necessários para dar suporte ao funcionamento dos aceleradores, obtendo-se os seguintes itens:

- Campo para identificação do acelerador, cada um possuindo um número de identificação único, possibilitando que a aplicação em software verifique os recursos atualmente configurados em hardware;
- Campo para versão do acelerador, possibilitando distinguir entre diferentes versões do mesmo acelerador e proporcionando compatibilidade entre aplicações;
- Comando para iniciar a execução do acelerador;
- Comando para *Reset* do acelerador para uma condição conhecida;

- Campo para informar estado atual do acelerador e informar como foi a execução;
- Campos para passar parâmetros e retornar resultados.

A interface de hardware AXI4-Lite Slave definida permite o uso de 16 registradores de 32-bits mapeados em memória, utilizando um endereço base de 32-bits e um *offset* de 8-bits para acessar cada registro, com o endereço físico dado pelas equações 3.1 e 3.2. O *offset* do registrador é dado pela sua posição de 0 à 15.

$$EndereçoRegistro = EndereçoBase + OffsetRegistro \tag{3.1}$$

$$OffsetRegistro = 4 * NúmeroRegistro \tag{3.2}$$

A partir dos itens levantados para dar suporte ao funcionamento dos aceleradores e da disponibilidade de registradores, define-se a seguinte interface de software:

- **Registrador de identificação**: Registro com *offset* 0x00, é permitido apenas a leitura do registro. É utilizado para a identificação do acelerador, seus campos são:

  31:16 *ID* (16-bits): Número único de identificação do acelerador, permite 65.536 identificadores distintos

  15:0 *Version* (16-bits): Versão do acelerador, permite 65.536 versões diferentes do mesmo acelerador

- **Registrador de comando e estado**: Registro com *offset* 0x04, é permitido a leitura e escrita do registro. É utilizado para comandar, verificar o estado atual e o resultado de uma operação do acelerador, seus campos são:

  31:8 *Reservado* (16-bits): Campo não utilizado, reservado para novas funcionalidades

  7:6 *Status* (2-bits): Estado atual do acelerador, podendo indicar estado ocioso (*idle*), executando (*running*), concluiu execução (*done*) ou erro na execução (*error*)

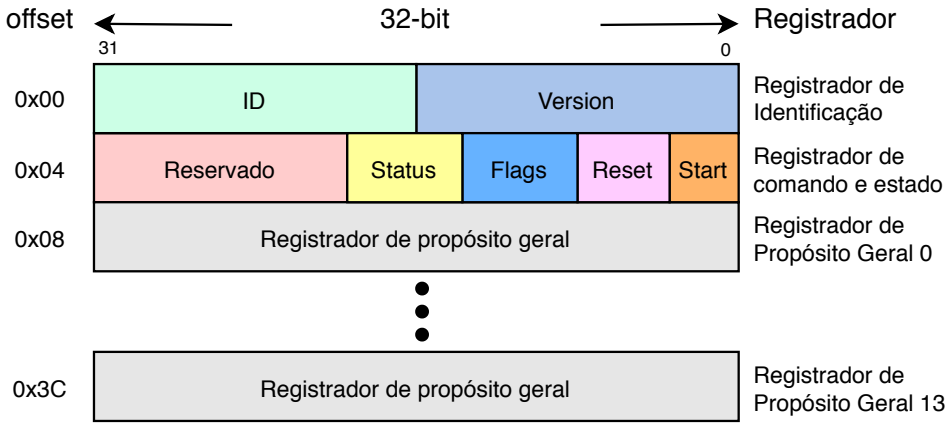
  5:2 *Flags* (4-bits): *Flags* de execução, podendo ser utilizado pelo desenvolvedor do acelerador para indicar alguma condição no final da execução

  1 *Reset* (1-bit): *Bit* auto-limpável para fazer a reinicialização do acelerador em um estado conhecido

  0 *Start* (1-bit): *Bit* auto-limpável para inicializar a operação do acelerador

- **Registradores de propósito geral**: Registradores de 0 à 13 com *offsets* 0x08 à 0x3C, respectivamente, que podem ser utilizados pelo desenvolvedor do acelerador para passar parâmetros, passar endereços de memória de vetores de dados, e receber resultados da execução

Figura 13 – Registradores do acelerador mapeados em memória e seus campos. Imagem ilustrativa, os campos não estão em escala em relação ao comprimento em *bits*.



Fonte: Produzido pelo autor

As definições dos registros podem ser observadas na Figura 13.

Definida a interface de software, parte-se então para a descrição e desenvolvimento da API.

3.2.2 API

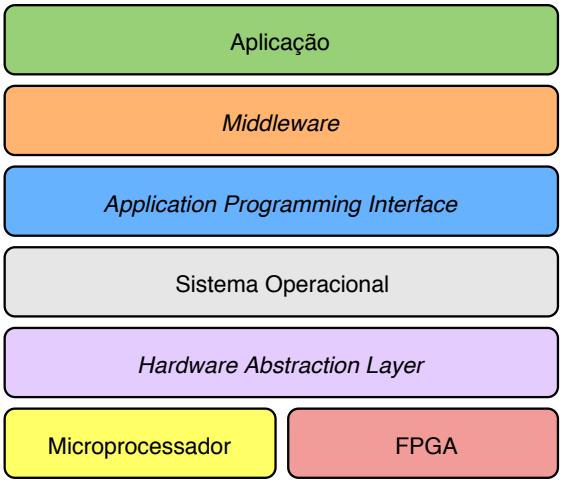
A API compõe uma camada acima do Sistema Operacional (SO), utilizando a padronização da interface de software definida anteriormente para implementar e dar suporte às suas funcionalidades.

Utilizando-se de chamadas de sistema do SO, a API consegue ter acesso aos registros dos aceleradores, fazer a reconfiguração total e parcial da malha lógica do FPGA, e alocar recursos de memória física contínua compartilhadas entre os aceleradores e o SO. As funcionalidades implementadas pela API são listadas abaixo, sendo elas dividias em três categorias:

- Reconfiguração:
  - Funções para reconfiguração total e reconfiguração parcial;
  - Escalonador para gerenciar quais MRs serão reconfigurados em uma chamada de reconfiguração.
- Alocação de recursos:
  - Alocação de memória física e contínua compartilhada entre o SO e os aceleradores.
- Controle:
  - Verificação de ID e versão dos aceleradores configurados nos MRs;

- Acesso de leitura e escrita aos registros de propósito geral;
- Verificação de estado do acelerador;
- Verificação de *flags* de execução;
- Comando para execução do acelerador;
- Comando para *reset* do acelerador.

Figura 14 – Camadas do sistema.



Fonte: Produzido pelo autor

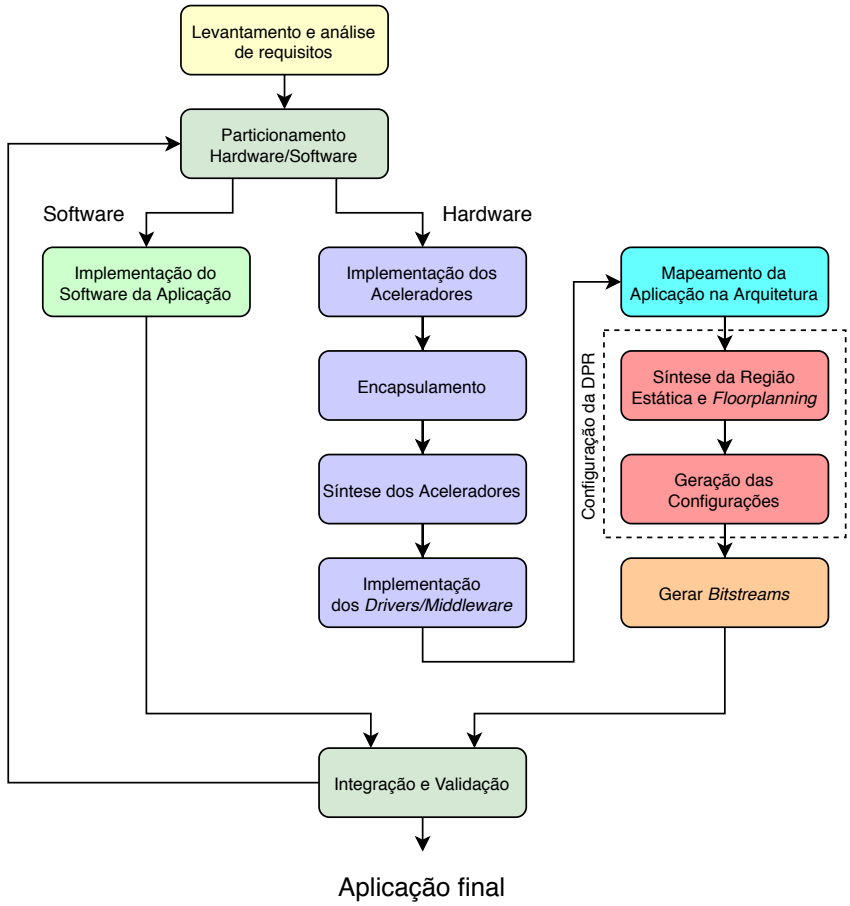
Utilizando as funcionalidades da API, é possível criar *drivers* para os aceleradores. Os *drivers* são implementações de alto nível que abstraem as chamadas da API para o acesso e controle dos aceleradores, possibilitando fazer chamadas para escalonar a configuração do acelerador em um dos MR disponível, alocando a quantidade de recursos de memória necessários para execução, formatando e passando os parâmetros para o acelerador, e permitindo a implementação de *callbacks* para informar a aplicação o fim da execução do acelerador. Os *drivers* criados para os aceleradores compõem a camada *Middleware*, acima da camada da API. A Figura 14 mostra a separação das camadas da aplicação.

3.3 Metodologia de Aplicação

Definidos os padrões das interfaces, a arquitetura de suporte e a API, vê-se então a necessidade de elaboração de uma metodologia para guiar desenvolvedores a utilizarem o particionamento com reconfiguração parcial e dinâmica em de sua aplicação. A Figura 15 mostra o diagrama de blocos da metodologia desenvolvida.

A metodologia possui as seguintes etapas:

Figura 15 – Metodologia de aplicação.



Fonte: Produzido pelo autor

1. Levantamento e Análise de Requisitos;
2. Particionamento Hardware/Software;
    3. Hardware:
        - 3.1 Implementação dos Aceleradores;
        - 3.2 Encapsulamento;
        - 3.3 Síntese dos Aceleradores;
        - 3.4 Implementação dos *Drivers*/*Middleware*;
        - 3.5 Mapeamento da Aplicação na Arquitetura;
        - 3.6 Síntese da Região Estática e *Floorplanning*;
        - 3.7 Geração das Configurações;
        - 3.8 Gerar *Bitstreams*;
    4. Software:

4.1 Implementação do Software da Aplicação;

5. Integração e Validação;

1. Levantamento e Análise de Requisitos. Na primeira etapa, como em qualquer outro projeto de sistema embarcado, são levantados os requisitos do sistema, assim como as funcionalidades necessárias, os dispositivos e plataformas disponíveis para o projeto e os recursos disponíveis para a implementação.

2. Particionamento Hardware/Software. Nessa etapa é necessário definir quais algoritmos e quais funcionalidades do sistema serão implementadas em hardware e quais serão implementadas em software. A escolha dependerá das restrições impostas à aplicação.

3.1 Implementação dos Aceleradores. Definido as funcionalidades que serão implementadas em hardware, faz-se então suas implementações. Para casos em que a funcionalidade já é disponibilizada por *Intellectual Property Cores* (*IP Cores*), não é necessário sua re-implementação.

3.2 Encapsulamento. Os aceleradores obtidos na etapa 3.1 são então encapsulados nas padronizações das interfaces de hardware e de software definidas, adequando-os à arquitetura de suporte e a API desenvolvida.

3.3 Síntese dos Aceleradores. Finalizado a etapa de encapsulamento, faz-se então a síntese individual dos aceleradores, gerando relatórios dos tipos e quantidade de recursos utilizados por cada um dos aceleradores sintetizados. Os tipos e quantidades de recursos utilizados pelos aceleradores serão utilizados na etapa de mapeamento da aplicação na arquitetura.

3.4 Implementação dos *Drivers*/*Middleware*. De acordo com a funcionalidade de cada um dos aceleradores, assim como a utilização de seus registros, modo de operação, parâmetros e *flags* de execução, e utilizando as funções disponíveis na API para acesso aos registros e comandos, são criados nessa etapa os *drivers* de cada um dos aceleradores, disponibilizando, para a camada de software, funções de alto nível que abstraem o funcionamento interno dos aceleradores. Os *drivers* compõem a camada de *Middleware* da aplicação e são responsáveis por abstrair as implementações em hardware dos aceleradores.

3.5 Mapeamento da Aplicação na Arquitetura. Nessa etapa deve ser considerado:

A. A quantidade e tipo de recursos utilizados por cada um dos aceleradores implementados, definindo o tamanho de cada acelerador;

B. A quantidade e tipo de recursos disponíveis na malha lógica disponível no dispositivo utilizado;

C. O padrão da interface entre o sistema de processamento e a malha do FPGA, definindo a necessidade ou não de um conversor de barramento para AMBA AXI4.

Primeiramente, define-se a região estática, sendo ela composta pelos interconectores AXI e, para o caso do item C, um conversor de barramento. A definição então da região reconfigurável

dependerá do tamanho do dispositivo (item B) e da quantidade de recursos utilizados pela região estática. Assim, com o tamanho definido da região reconfigurável, a quantidade de MRs possíveis mapeados na arquitetura de suporte dependerá do tamanho do maior acelerador implementado (item A).

3.6 Síntese da Região Estática e *Floorplanning*. Definido a região estática e os módulos reconfiguráveis, faz-se então a síntese da região estática e o *floorplanning* manual da malha do FPGA.

O *floorplanning* é o processo onde a malha do FPGA é dividia em áreas que serão estáticas e em áreas que serão reconfiguráveis, sendo as áreas reconfiguráveis ocupadas pelos MRs. É uma etapa obrigatória para a utilização da técnica de reconfiguração parcial e dinâmica e os detalhes do processo variam por dispositivo e ferramenta.

3.7 Geração das Configurações. Feito o *floorplanning* com as divisões das regiões dos MRs e região estática, faz-se então a geração das configurações. As configurações são a disposição dos aceleradores nos módulos reconfiguráveis que os suportam, ou seja, para cada combinação possível de aceleradores configurados nos MRs é gerado uma configuração.

Considere o exemplo onde tenha três aceleradores distintos (A, B e C) e dois MRs (1 e 2) disponíveis, considerando que o MR 1 suporta apenas os aceleradores A e B, e o MR 2 suporta todos os três aceleradores A, B e C, tem-se, então, um total de seis configurações, como mostra a Tabela 1. Esta etapa também é obrigatória para utilização da técnica de reconfiguração parcial e dinâmica.

Tabela 1 – Configurações possíveis para o exemplo

Configuração	MR 1	MR 2
1	A	A
2	A	B
3	A	C
4	B	A
5	B	B
6	B	C

3.8 Gerar *Bitstreams*. Nessa etapa é feito o processo de geração das *bitstreams*, gerando-se uma *bitstream* parcial para cada configuração de acelerador, ou seja, uma para cada MR em que o acelerador possa ser configurado, e uma *bitstream* total da região estática, tendo-se assim o conjunto de *bitstreams* que serão utilizadas na aplicação.

4.1 Implementação do Software da Aplicação. Nessa etapa é desenvolvido o software da aplicação que será executado pelo sistema de processamento, utilizando os *drivers* presentes na camada de *middleware* para utilizar os aceleradores implementadas no hardware. As etapas de desenvolvimento de software ocorrem paralelas às etapas de desenvolvimento de hardware.

5. Integração e Validação. Nessa etapa o software da aplicação e os aceleradores desenvolvidos são integrados na aplicação e testes são feitos para validar o funcionamento e se a

implementação atende as restrições impostas à aplicação. Caso os requisitos não sejam atendidos, volta-se à etapa 2 de particionamento e um novo particionamento é feito. Caso o sistema seja validado, ele é implantado na aplicação final.

A metodologia apresentada nesse capítulo será aplicada em um estudo de caso, no capítulo 4, para que se possa avaliar sua adequação a um projeto real.

4 Estudo de Caso: Implementação de um Classificador

Como estudo de caso, utilizou-se o classificador NN-clas proposto por Gade (2018) para validar o método proposto.

As próximas seções descrevem-se as etapas de execução do classificador, o ambiente de desenvolvimento utilizado, a implementação do classificador utilizando o método de particionamento proposto, e, em seguida, apresentam-se os resultados obtidos e uma discussão.

4.1 Classificador NN-clas

A execução do classificador é dividia em duas etapas, sendo as etapas de treinamento e classificação.

A etapa de treinamento é a primeira etapa executada, onde as amostras de treinamento são utilizadas para gerar o grafo de Gabriel e, posteriormente, obter as arestas de suporte utilizadas na classificação. O grafo de Gabriel é gerado inicialmente calculando-se a distância ponto a ponto entre todas as amostras de treinamento, podendo-se utilizar métricas como, por exemplo, a distância euclidiana (Equação 2.2), a distância de Manhattan (Equação 4.1) ou a distância euclidiana quadrática (Equação 4.2).

$$D(X,Y) = \sum_{n=1}^{M} |x_i - y_i| \tag{4.1}$$

$$D(X,Y) = \sum_{n=1}^{M} (x_i - y_i)^2 \tag{4.2}$$

Após calcular-se as distâncias, verifica-se a condição de aresta para todas as amostras, formando-se arestas apenas os pares de amostras que satisfaçam a condição da Equação 2.1. Do conjunto de arestas, obtêm-se e armazena-se as arestas de suporte, sendo as arestas no qual os seus vértices pertencem a classes distintas. Desta forma, a etapa de treinamento é subdividida nas seguintes etapas:

- Cálculo da distância entre as amostras de treinamento;
- Cálculo das arestas do grafo de Gabriel;
- Verificação das arestas de suporte;
- Armazenar arestas de suporte.

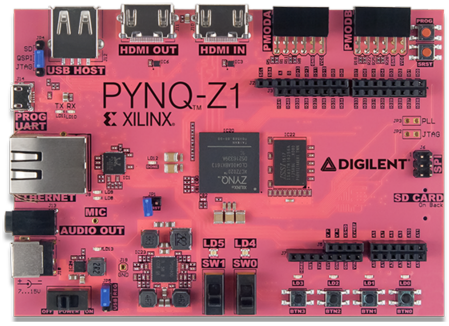
A etapa de classificação utiliza das arestas de suporte obtidas no treinamento para classificar as amostras de teste. Para classificar a amostra, o critério do NN-clas utiliza as distâncias entre a amostra de teste e os vértices das arestas de suporte, sendo a amostra de teste classificada com a mesma classe do vértice mais próximo a ela, ou seja, com a menor distância. Desta forma, a etapa de classificação é subdivida nas seguintes etapas:

- Cálculo da distância entre as amostras de teste e os vértices das arestas de suporte;
- Verificação do vértice mais próximo (menor distância).

4.2 Ambiente de Desenvolvimento

Para a implementação do estudo de caso proposto, optou-se pela utilização do kit de desenvolvimento PYNQ-Z1, que contém o SoC FPGA *Zynq-7000* fabricado pela empresa Xilinx. O kit pode ser observado na Figura 16.

Figura 16 – Kit de desenvolvimento PYNQ-Z1.

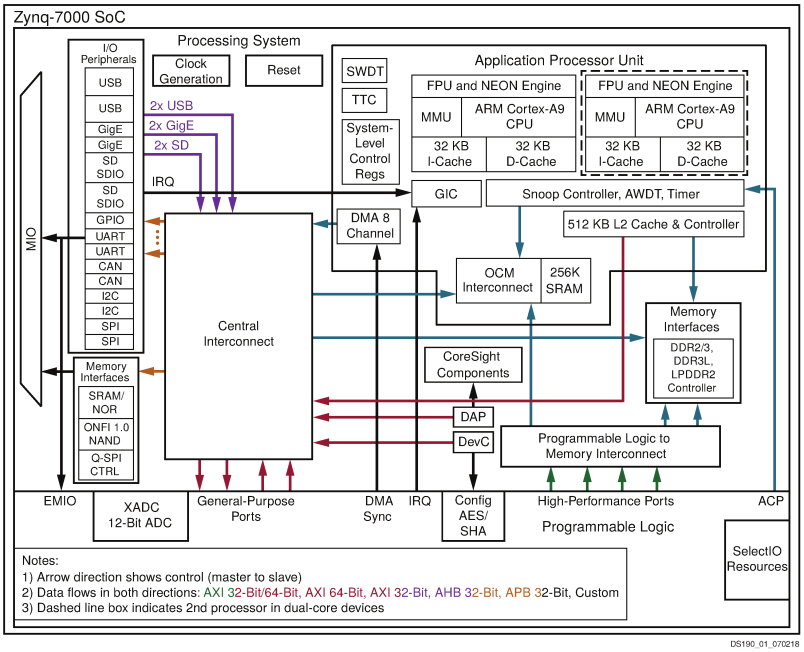


Fonte: Digilent (2019)

O *Zynq-7000*, podendo sua arquitetura ser observada na Figura 17, possui um processador *dual-core* ARM Cortex-A9, e vários periféricos, destacando-se os periféricos de USB, Gigabit Ethernet e UART, além de possuir um controlador de memória externa integrado e uma malha lógica que, para o dispositivo Z-7020 presente no kit, dispõe de:

- 85K de células lógicas programáveis;
- 53.2K de LUTs;
- 106.4k de FFs;
- 4.9 Mb de blocos de memória RAM (*Random-Access Memory*);
- 220 blocos de DSP.

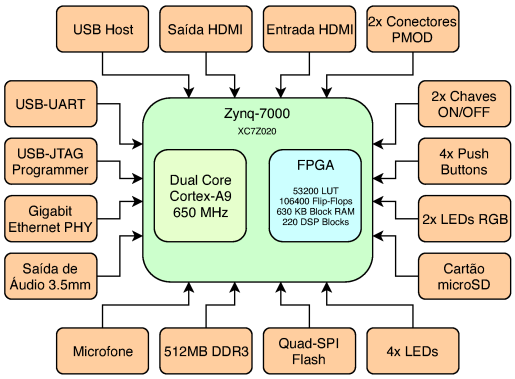
Figura 17 – Arquitetura interna do Zynq-7000



Fonte: Xilinx (2018)

O diagrama de blocos da Figura 18 mostra as interfaces e os recursos disponibilizados no kit, sendo as interface de rede *Gigabit Ethernet* e cartão *microSD* as interfaces físicas externas necessárias para esse estudo de caso.

Figura 18 – Diagrama de blocos do kit de desenvolvimento PYNQ-Z1.



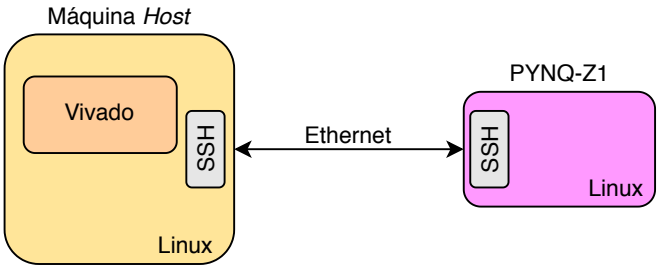
Fonte: Produzido pelo autor

Para o desenvolvimento dos acelerados opta-se pela utilização do software Vivado da pela empresa Xilinx, devido ao suporte fornecido para o dispositivo *Zynq-7000* presente no kit de desenvolvimento. O processador ARM presente no dispositivo possibilita a execução do sistema operacional Linux, sendo ele então utilizado no dispositivo para gerenciar os periféricos

e recursos, e executar a camada de software da aplicação.

Utiliza-se uma máquina *host* com o sistema operacional Linux para o desenvolvimento do software da aplicação na linguagem C e os aceleradores no software Vivado utilizando a linguagem Verilog HDL. A máquina *host* se conecta ao kit através da interface de rede utilizando um terminal de *Security SHell* (SSH). A Figura 19 ilustra o ambiente de desenvolvimento utilizado.

Figura 19 – Ambiente de desenvolvimento.



Fonte: Produzido pelo autor

4.3 Implementação

Seguindo a metodologia desenvolvida, primeiramente faz-se, para a implementação do classificador, o levantamento e a análise dos requisitos e as funcionalidades desejadas.

Para o estudo de caso proposto, deseja-se que o classificador dê suporte a amostras com até 32 características, sendo a representação numérica de cada característica feita em ponto flutuante de precisão simples (32-bits) de acordo com o padrão IEEE-754 (IEEE, 2008), e dê suporte à conjuntos de treinamentos com o número mínimo de 1024 amostras de treinamento por conjunto. O sistema deverá ser capaz de buscar as amostras de treinamento e classificação em arquivos de texto formatados, e sua execução seja feita em lotes, ou seja, busque as amostras de treinamento do lote, faça o treinamento do classificador, busque as amostras de teste do lote, e faça a classificação das amostras.

A execução do classificador requer que as etapas que compõe as fases de treinamento e classificação sejam executadas de forma sequencial, sendo as etapas enumeradas da seguinte forma:

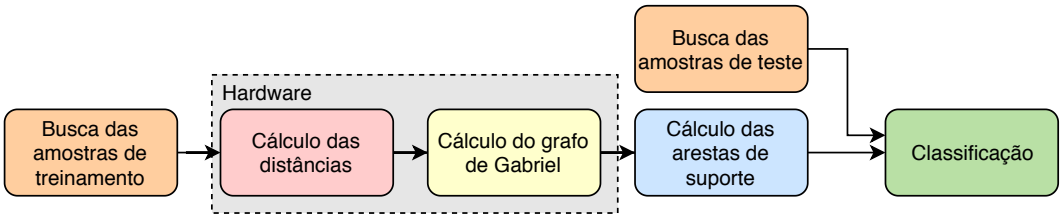
1. Busca das amostras de treinamento;
2. Cálculo das distâncias entre as amostras de treinamento;
3. Cálculo do grafo de Gabriel;
4. Cálculo das arestas de suporte;

5. Busca das amostras de teste;

6. Classificação das amostras de teste.

Analisando-se as etapas, optou-se, para esse estudo de caso, por implementar aceleradores em hardware para as etapas 2, utilizado-se a métrica de Manhattan, e 3, visto que esses algoritmos são propícios a explorar o paralelismo da execução em hardware e compõe a fase de treinamento, onde o maior volume de amostras são processadas. Desta forma, os algoritmos das etapas 4 e 6, junto com as etapas 1 e 5 responsáveis por alimentar o classificador com os lotes de dados, são implementadas em software. A Figura 20 mostra visualmente o fluxo das etapas executadas pelo classificador.

Figura 20 – Etapas do NN-clas.



Fonte: Produzido pelo autor

Definidas as etapas e os algoritmos a serem aceleradas em hardware, faz-se então os processos de implementação, validação (utilizando simulações), encapsulamento, síntese, e implementação dos *drivers* para cada um dos aceleradores. A Tabela 2 mostra os tipos e quantidades de recursos utilizados pelos aceleradores implementados.

Tabela 2 – Recursos utilizados pelos aceleradores

Acelerador	Recursos		
	LUT	FF	DSP
Distância de Manhattan	25990	20062	0
Grafo de Gabriel	8751	9669	96

Após a síntese e utilizando-se o relatório dos tipos e quantidades de recursos utilizados pelos aceleradores, faz-se as definições de arquitetura.

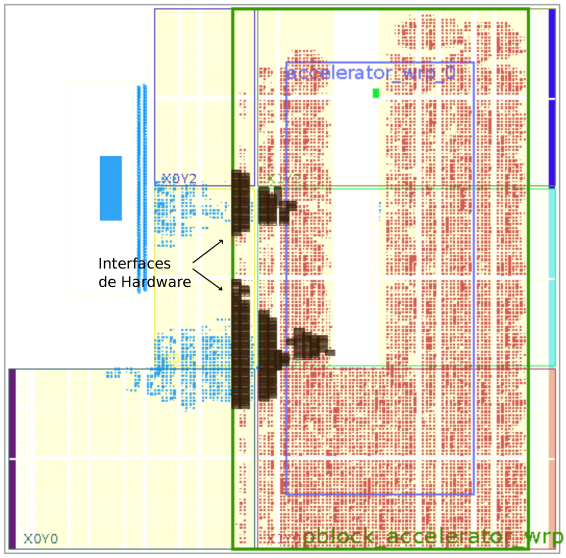
O dispositivo Zynq-7000 utiliza a interface AMBA AXI4 para conectar o sistema de processamento à malha lógica do FPGA, não sendo então necessário utilizar um conversor de barramento, conectando o sistema de processamento diretamente aos interconectores AXI presentes na região estática.

Analisando-se os recursos utilizados pelos os aceleradores e a quantidade de recursos disponíveis no dispositivo, implementa-se apenas um módulo reconfigurável.

Após o mapeamento da aplicação na arquitetura, faz-se então o *floorplanning* e sintetiza-se a região estática, e, em seguida, geram-se as configurações.

Com dois aceleradores e apenas um MR disponível, tem-se duas configurações possíveis, uma para cada acelerador ocupando o MR. O *floorplanning* para as duas configurações possíveis podem ser visto na Figura 21, para o acelerador da distância de Manhattan, e na Figura 22, para o acelerador do grafo de Gabriel. Nas Figuras 21 e 22, observa-se a região reconfigurável do MR delimitado por um retângulo verde e os recursos destacados na malha são as conexões das interfaces de hardware do acelerador com a região estática e a distribuição em forma de colunas dos blocos de DSPs na arquitetura do dispositivo.

Figura 21 – *Floorplanning* da configuração única do acelerador da distância de Manhattan.



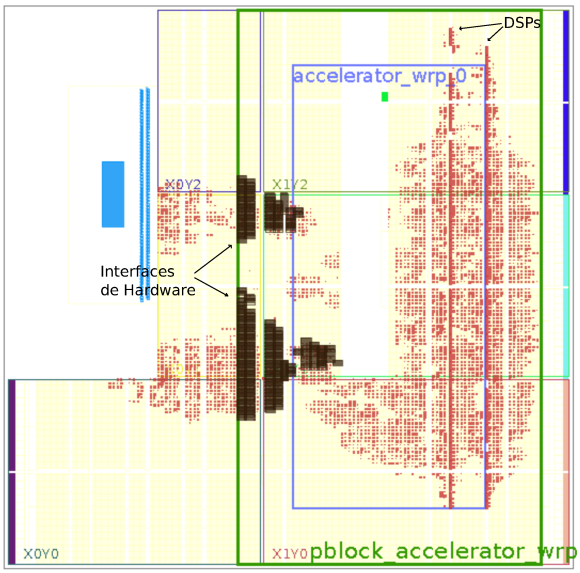
Fonte: Produzido pelo autor

Com as configurações geradas e mapeadas, geram-se as *bitstreams* parciais para os aceleradores e a *bitstream* total para a região estática, sendo elas, junto com os *drivers* desenvolvidos, utilizadas na integração dos aceleradores com o software da aplicação.

4.4 Testes de Desempenho

Para a avaliação do impacto do particionamento hardware/software no desempenho do classificador, utilizou-se de execuções completas dos algoritmos em software como referência, sendo executados no processador do dispositivo SoC FPGA, e particionamentos utilizando os aceleradores implementados na malha lógica do dispositivo. Desta forma, avaliou-se o tempo de execução do classificador para os seguintes particionamentos:

Figura 22 – *Floorplanning* da configuração única do acelerador do grafo de Gabriel.



Fonte: Produzido pelo autor

1: Execução completa em software, sem particionamento, todas as etapas de treinamento e classificação executadas em software;

2: Particionamento utilizando o acelerador da distância de Manhattan em hardware e o restante das etapas em software;

3: Particionamento utilizando o acelerador do cálculo do grafo de Gabriel em hardware e o restante das etapas em software;

4: Particionamento com os aceleradores da distância de Manhattan e do cálculo do grafo de Gabriel em hardware utilizando reconfiguração parcial e dinâmica, e o restante das etapas em software.

Utilizou-se na avaliação uma base de dados de amostras com 32 características, cada característica representada em ponto flutuante de precisão simples (32-bits), geradas de forma aleatória e não permitindo a ocorrência de sobreposição de classes. Testou-se o classificador com lotes contendo 1024 amostras de treinamento e 64 amostras de teste da base de dados.

Utilizando-se a execução completa em software (configuração 1) como referência, observam-se os tempos obtidos para as execuções na Tabela 3, sendo utilizado a ferramenta *time* do Linux para fazer as medições.

O tempo de execução da configuração 4 requer que os aceleradores sejam reconfigurados durante a execução do classificador, e o tempo gasto observado para o processo de reconfiguração parcial foi de $54,7$ ms (já incluído no valor apresentado na tabela).

Tabela 3 – Tempos de execução para os diferentes testes de particionamentos hardware/software.

Tipo de execução	Tempo de execução (s)	Redução (%)
Configuração 1	221,79 s	-
Configuração 2	213,81 s	-3,6%
Configuração 3	145,37 s	-34,5%
Configuração 4	137,72 s	-37,9%

4.5 Discussão e Resultados

A padronização das interfaces de hardware dos aceleradores utilizando as interfaces AMBA AXI4 Master e AMBA AXI4-Lite Slave, permitiu que os aceleradores do classificador acessassem a memória externa para buscar e armazenar, de forma rápida e direta, dados na memória, além de disponibilizar uma interface de comunicação entre o acelerador e o sistema de processamento, permitindo o acesso e controle do acelerador pelo sistema de processamento.

A utilização da arquitetura de proposta possibilitou que ambos os aceleradores pudessem ser configurados na mesma região do FPGA de forma intercalada no tempo, utilizando a reconfiguração parcial e dinâmica. Verifica-se que a distribuição não homogênea dos diferentes tipos de recursos do FPGA afeta a divisão das regiões para os MRs criando regiões que podem possuir recursos subutilizados de acordo com o acelerador que se encontra em atividade. Para ilustrar essa situação, observe que o acelerador para o cálculo do grafo de Gabriel necessita de muitos blocos de DSPs e esses recursos são dispostos na malha do FPGA em colunas (Figura 22). Apesar de o acelerador de Distância Manhattan não utilizar tais blocos, como os dois aceleradores utilizam a mesma região, foi necessário que se considerasse uma área que possuísse recursos suficientes para atender o bloco com a maior necessidade dessas estruturas.

A divisão das regiões para os MRs depende também das limitações impostas pelas ferramentas de projeto e do próprio dispositivo utilizado, como é o caso do dispositivo Zynq e a ferramenta de desenvolvimento Vivado, que limitam a divisão das regiões reconfiguráveis à retângulos, aumentando a sub-utilização dos recursos na malha do FPGA.

A adequação das interfaces de software dos aceleradores desenvolvidos para o classificador ao padrão proposto permitiu que os aceleradores fossem acessados por funções já implementadas na API proposta, facilitando o controle e a passagem de parâmetros, como por exemplo o ponteiro do vetor com as amostras de treinamento e o número de amostras, da aplicação de software para os aceleradores em hardware, sem que o desenvolvedor da aplicação sequer conheça o protocolo de comunicação ou mesmo a interface ao qual os aceleradores se conectam ao sistema de processamento. Assim, utilizando as funções da API em conjunto com a padronização das interfaces de software, foram possíveis as implementações dos *drivers* para os aceleradores do classificador, possibilitando que a aplicação de software do classificador consiga utilizar os recursos de hardware de forma completamente transparente e que ela fosse desenvolvida independente dos aceleradores em hardware.

Os testes de desempenho mostraram os ganhos obtidos ao se utilizar diferentes particionamentos hardware/software para a aplicação, com o maior ganho de desempenho obtido quando é feito particionamento com reconfiguração parcial e dinâmica dos aceleradores, que, apesar de gastar cerca de $54,7$ ms no processo de reconfiguração, obtém-se uma redução de 37,9% do tempo de execução ao se comparar com a execução total em software, sem particionamento. Vale observar também a diferença de ganho de desempenho entre os dois aceleradores desenvolvidos, verificando-se que o acelerador para o cálculo do grafo de Gabriel representa um maior ganho para a aplicação do classificador do que o acelerador para o cálculo da distância.

Observa-se que a metodologia para aplicação do método proposto pôde ser utilizada com sucesso para guiar a implementação do estudo de caso.

5 Conclusão

Esse trabalho apresentou um método para particionamento hardware/software utilizando reconfiguração parcial e dinâmica.

A padronização das interfaces de hardware dos aceleradores utilizando as interfaces AMBA AXI4 foi capaz de atender os requisitos de comunicação necessários para a utilização dos aceleradores em aplicações reais com particionamento hardware/software, assim como foi mostrado no estudo de caso, satisfazendo o item A dos objetivos específicos.

A arquitetura de suporte proposta, dividindo a malha lógica do FPGA em regiões estáticas e reconfiguráveis e utilizando de módulos reconfiguráveis como receptáculos para os aceleradores, mostrou-se capaz de atender os requisitos necessários para a utilização de reconfiguração parcial e dinâmica, assim como foi mostrado no estudo de caso apresentado, satisfazendo o item B dos objetivos específicos.

As padronizações propostas para as interfaces de software dos aceleradores, possibilitaram o desenvolvimento e utilização de uma API para fornecer funções de acesso, controle e reconfiguração dos aceleradores implementados em hardware, possibilitando a criação de *drivers* para os aceleradores e permitindo a utilização dos mesmos de forma transparente para as aplicações em software, conforme foi mostrado no estudo de caso apresentado, satisfazendo os itens C e D dos objetivos específicos.

O estudo de caso apresentado foi desenvolvido utilizando a metodologia proposta para a aplicação do método, e, com o sucesso da implementação do classificador do estudo de caso, mostra que a metodologia foi capaz de guiar o desenvolvimento, satisfazendo o item E dos objetivos específicos.

O método para particionamento hardware/software utilizando reconfiguração parcial e dinâmica proposto mostrou-se capaz de particionar aplicações utilizando co-design hardware/software, auxiliando na etapa de integração e evitando o acoplamento das camadas de hardware e software da aplicação, sendo o método utilizado com sucesso no particionamento do estudo de caso.

Desta forma, mostra-se que os objetivos específicos e gerais foram atendidos considerando sua aplicação no estudo de caso.

São apontados como trabalhos futuros o estudo e desenvolvimento de ferramentas para auxiliar no processo de *floorplanning*, processo que nas ferramentas atuais disponíveis ainda é feito de forma manual, e desenvolvimento de ferramentas para otimização da distribuição das regiões reconfiguráveis, permitindo uma melhor utilização da malha lógica, reduzindo a perda por fragmentação.

Referências

AFONSO, G. et al. Heterogeneous CPU/FPGA Reconfigurable Computing System for Avionic Test Application. In: *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*. [S.l.: s.n.], 2013. ISSN null. Citado na página 32.

ARM. *AMBA 4 AXI4-Stream Protocol*. [S.l.], 2010. Version: 1.0. Citado na página 34.

ARM. *AMBA AXI and ACE Protocol Specification*. [S.l.], 2011. Issue D. Citado na página 33.

DIGILENT. *Digilent Resource Center Webpage*. 2019. Disponível em: <https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/start>. Citado na página 54.

GADE, L. dos R. *Estudo e Desenvolvimento Arquitetural para Implementação de um Classificador Geométrico de Margem Larga em Sistemas Embarcados*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais, 2018. Citado 3 vezes nas páginas 35, 36 e 53.

GOVIL, N.; CHOWDHURY, S. R. High Performance and Low Cost Implementation of Fast Fourier Transform Algorithm Based on Hardware Software Co-Design. In: *2014 IEEE REGION 10 SYMPOSIUM*. [S.l.: s.n.], 2014. p. 403–407. Citado na página 38.

IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, p. 1–70, Aug 2008. Citado na página 56.

INTEL. *Intel® Quartus® Prime Pro Edition User Guide: Partial Reconfiguration*. [S.l.], 2019. UG-20136. Citado na página 32.

JAMES, G. et al. *An introduction to statistical learning*. [S.l.]: Springer, 2013. v. 112. Citado 2 vezes nas páginas 35 e 36.

KELM, J. H.; LUMETTA, S. S. HybridOS: Runtime Support for Reconfigurable Accelerators. In: *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*. New York, NY, USA: ACM, 2008. (FPGA '08), p. 212–221. ISBN 978-1-59593-934-0. Disponível em: <http://doi.acm.org/10.1145/1344671.1344703>. Citado na página 38.

KRILL, B. et al. An Efficient FPGA-Based Dynamic Partial Reconfiguration Design Flow and Environment for Image and Signal Processing IP Cores. *Signal Processing: Image Communication*, v. 25, n. 5, p. 377 – 387, 2010. ISSN 0923-5965. Special Issue on Breakthrough Hardware Architectures. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0923596510000494>. Citado na página 25.

LAHTI, S. et al. Are We There Yet? A Study on the State of High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 38, n. 5, p. 898–911, May 2019. ISSN 1937-4151. Citado na página 31.

LIE, W.; FENG-YAN, W. Dynamic Partial Reconfiguration in FPGAs. In: *2009 Third International Symposium on Intelligent Information Technology Application*. [S.l.: s.n.], 2009. v. 2, p. 445–448. Citado na página 25.

MAXFIELD, C. *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*. [S.l.]: Elsevier Science, 2004. ISBN 9780080477138. Citado na página 31.

MEYER-BAESE, U. *Digital Signal Processing with Field Programmable Gate Arrays*. [S.l.]: Springer Berlin Heidelberg, 2013. (Signals and Communication Technology). ISBN 9783662067284. Citado na página 30.

MITCHELL, T. M. *Machine Learning*. [S.l.]: McGraw-Hill Education, 1997. ISBN 0070428077. Citado na página 35.

MOURADIAN, C. et al. A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, v. 20, n. 1, p. 416–464, Firstquarter 2018. Citado na página 23.

NGO, H. T. et al. Real-Time Video Surveillance on an Embedded, Programmable Platform. *Microprocess. Microsyst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 37, n. 6-7, p. 562–571, ago. 2013. ISSN 0141-9331. Disponível em: <http://dx.doi.org/10.1016/j.micpro.2013.06.003>. Citado na página 38.

SASS, R.; SCHMIDT, A. *Embedded Systems Design with Platform FPGAs: Principles and Practices*. [S.l.]: Elsevier Science, 2010. ISBN 9780080921785. Citado 2 vezes nas páginas 29 e 30.

SHI, W. et al. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, v. 3, n. 5, p. 637–646, Oct 2016. Citado na página 23.

SODHRO, A. H.; PIRBHULAL, S.; ALBUQUERQUE, V. H. C. de. Artificial Intelligence-Driven Mechanism for Edge Computing-Based Industrial Applications. *IEEE Transactions on Industrial Informatics*, v. 15, n. 7, p. 4235–4243, July 2019. Citado na página 23.

STITT, G.; LYSECKY, R.; VAHID, F. Dynamic Hardware/Software Partitioning: A First Approach. In: *Proceedings of the 40th Annual Design Automation Conference*. New York, NY, USA: ACM, 2003. (DAC '03), p. 250–255. ISBN 1-58113-688-9. Citado na página 24.

TEICH, J. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, v. 100, n. Special Centennial Issue, p. 1411–1430, May 2012. ISSN 1558-2256. Citado na página 24.

TORRES, L. C. B. *Classificador por Arestas de Suporte (CLAS): Métodos de Aprendizado Baseados em Grafos de Gabriel*. Tese (Doutorado) — Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais, 2016. Citado 2 vezes nas páginas 36 e 37.

VATSOLAKIS, C.; PNEVMATIKATOS, D. RACOS: Transparent Access and Virtualization of Reconfigurable Hardware Accelerators. In: *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. [S.l.: s.n.], 2017. p. 11–19. Citado na página 38.

VIPIN, K.; FAHMY, S. A. FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 4, p. 72:1–72:39, jul. 2018. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3193827>. Citado 2 vezes nas páginas 25 e 26.

XILINX. *Zynq-7000 SoC Data Sheet: Overview*. [S.l.], 2018. Citado na página 55.

XILINX. *Vivado Design Suite User Guide: Partial Reconfiguration*. [S.l.], 2019. UG909 (v2019.1). Citado na página 32.

XU, S.; SCHAFER, B. C. Approximate Reconfigurable Hardware Accelerator: Adapting the Micro-Architecture to Dynamic Workloads. In: *2017 IEEE International Conference on Computer Design (ICCD)*. [S.l.: s.n.], 2017. p. 113–120. Citado na página 38.

YOON, I.; JOUNG, H.; LEE, J. Zynq-Based Reconfigurable System for Real-Time Edge Detection of Noisy Video Sequences. *Journal of Sensors*, v. 2016, p. 1–9, 08 2016. Citado na página 38.