UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ÉRICO DE MORAIS NUNES

Uma plataforma para agentes em hardware utilizando reconfiguração parcial

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Carlos Eduardo Pereira

Porto Alegre
2018

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Nunes, Érico de Morais

Uma plataforma para agentes em hardware utilizando reconfiguração parcial / Érico de Morais Nunes. – Porto Alegre: PPGC da UFRGS, 2018.

108 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Orientador: Carlos Eduardo Pereira.

1. Sistemas Embarcados. 2. Sistemas Multi-Agente. 3. Reconfiguração parcial de FPGA. I. Pereira, Carlos Eduardo. II. Título.

AGRADECIMENTOS

À Universidade Federal do Rio Grande do Sul e ao PPGC por me prover ensino público e de qualidade durante o período de aluno especial e o mestrado.

Ao professor Carlos Eduardo Pereira pela disponibilidade para me orientar e pela confiança em me aceitar como aluno do PPGC, mesmo como aluno em tempo parcial.

Ao colega Lucas Behnck pelos trabalhos em conjunto, por ajudar nas definições e trabalhos nos estudos de caso.

Ao colega Dionísio Doering pelo apoio com acesso ao hardware fundamental para o desenvolvimento do trabalho.

Ao colega Fábio Celestino Pereira pelo trabalho em conjunto e uma primeira oportunidade de aplicar os resultados iniciais deste trabalho.

À DATACOM por incentivar os estudos de pós-graduação com período de redução de carga horária.

RESUMO

Este trabalho apresenta o projeto e arquitetura de uma plataforma para execução de Agentes com funções implementadas em hardware, tomando vantagem do uso de hardware reconfigurável. Os Agentes em hardware são implementados utilizando dispositivos FPGA (Field-programmable Gate Array). O trabalho estende trabalhos anteriores semelhantes na área, com o diferencial de adicionar suporte às funcionalidades de reconfiguração parcial do hardware, suportar aplicações que demandam alto desempenho em hardware – como processamento de sinais e imagens – e redução de recursos de hardware necessários para execução da interface em software.

A plataforma proposta utiliza o framework JADE (Java Agent Development Framework), que é um dos frameworks mais populares no estado da arte de desenvolvimento de Agentes e compatível com outros frameworks de Agentes através da conformidade aos padrões FIPA (Foundation for Intelligent Physical Agents). Com o uso do JADE, a plataforma possibilita a comunicação entre Agentes com funções implementadas em hardware e Agentes puramente implementados em software dentro de um mesmo SMA (Sistema Multi-Agente).

Uma funcionalidade notável do JADE é a possibilidade de migração de Agentes entre plataformas de um mesmo SMA. Através do uso da reconfiguração parcial de hardware em conjunto com o JADE, a plataforma permite a migração de Agentes de software para hardware e vice-versa, além de suportar reconfiguração de múltiplos Agentes em hardware com um único FPGA.

A plataforma faz uso de um único chip através do uso de um processador soft core implementado na lógica programável. O uso deste processador é um diferencial neste trabalho, e mostra que é possível utilizar o JADE em sistemas embarcados com recursos de processamento limitados. Ou seja, em um Agente cuja principal função é implementada em hardware, basta um processador bastante simples para atuar como uma interface entre o hardware e o framework de Agentes. O uso do processador dentro do FPGA tem também o benefício de oferecer formas de acesso mais integrado ao hardware, permitindo maior desempenho na transmissão de dados ao hardware.

A plataforma foi validada através de estudos de caso de Agentes com implementações em hardware e em software, incluindo um estudo de caso aplicado de processamento de imagem embarcado utilizando VANTs (Veículos Aéreos Não-Tripulados). O estudo também apresenta comparações de desempenho entre a execução dos Agentes em hardware e em

outras plataformas embarcadas de prateleira. Os experimentos realizados mostram um ganho significativo de desempenho nas implementações em FPGA, especialmente considerando processamento de imagens de alta resolução, mesmo considerando que o FPGA executa em frequências consideravelmente reduzidas em comparação às outras plataformas testadas.

Palavras-chave: Sistemas Embarcados. Sistemas Multi-Agente. Reconfiguração parcial de FPGA.

A platform for hardware agents using partial reconfiguration

ABSTRACT

This work described the design and architecture of a platform for execution of Agents whose functions are implemented in hardware, by leveraging the use of reconfigurable hardware. The hardware Agents are implemented using FPGA (Field-programmable Gate Array) devices. This work extends previous similar work in this field, while adding the features of hardware partial reconfiguration, supporting applications which require high performance in hardware – such as image or signal processing – and reducing the hardware resource for the software interface execution.

The proposed platform makes use of the JADE (Java Agent Development Framework) framework, which is one of the most popular frameworks in state-of-the-art Agent development, and is also compatible with other Agent development frameworks due to compliance with FIPA (Foundation for Intelligent Physical Agents) standards. With the use of JADE, the platform enables communication among Agents which are implemented in hardware and Agents purely implemented in software, inside the same MAS (Multi-Agent System).

One notable feature of JADE is the possibility of migrating Agents among platforms inside a single MAS. Through the use of hardware partial reconfiguration along with JADE, the platform enables the migration of Agents from software to hardware and vice-versa, in addition to supporting múltiple hardware Agents in a single FPGA.

The platform makes use of a single chip, by using a MicroBlaze soft core processor implemented in programmable logic. The use of this processor is a distinction on this work, and it shows that it is possible to use JADE on embedded systems with limited processing power. That is, in an Agent whose main function is implemented in hardware, a very simple processor to act as an interface between hardware and the Agent framework is enough. The use of the soft core processor inside the FPGA also has the benefit of offering more integrated ways of accessing hardware, enabling higher performance for transferring data to hardware.

The platform was validated through case studies of hardware and software Agent implementation, including a case study applied to image processing using UAVs (Unmanned Aerial Vehicles). The study also shows performance comparisons between the Agent execution in hardware and in other off-the-shelf embedded platforms. The performed

experiments report a significative performance increase in the FPGA implementations, particularly in high resolution image processing, even considering that the FPGA runs in considerably lower clock frequency than the other tested platforms.

Keywords: Embedded Systems, Multi-agent Systems, Partial FPGA Reconfiguration.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACL Agent Communication Language.

AMBA Advanced Microcontroller Bus Architecture.

AMS Agent Management System.

API Application Programming Interface.

ASIC Application-specific integrated circuit.

AXI Advanced eXtensible Interface.

BRAM Block Ram.

CMA Contiguous Memory Allocator.

CPLD Complex programmable logic device.

CPU Central Processing Unit.

DF Directory Facilitator.

DMA Direct Memory Access.

DSP Digital Signal Processor.

DTB Device Tree Blob.

DTS Device Tree Source.

EDA Electronic Design Automation.

EDK Embedded Development Kit.

FDT Flattened Device Tree.

FIPA Foundation for Intelligent Physical Agents.

FIR Finite Impulse Response.

FPGA Field-programmable Gate Array.

FPU Floating-point unit.

GPL General Public License.

GUI Graphical user interface.

HDL Hardware Description Language.

HLS High-level synthesis.

HWICAP Hardware Internal Configuration Access Port.

IOT Internet of Things.

JADE Java Agent Development Framework.

JADE-LEAP Java Agent Development Framework-Lightweight Extensible Agent Platform.

JTAG Joint Test Action Group.

JVM Java Virtual Machine.

LGPL Lesser General Public License.

LUT Lookup Table.

MAS Multi-Agent System.

MMIO Memory Mapped Input/Output.

MMU Memory Management Unit.

PIO Programmed Input/Output.

PLC programmable logic controller.

POSIX Portable Operating System Interface.

RMA Remote Management Agent.

RSSF Redes de Sensores Sem Fio.

RTSJ Real-Time Specification for Java.

SBC Single-board computer.

SIMD Single instruction, multiple data.

SMA Sistema Multi-Agente.

SO Sistema Operacional.

SOC System on Chip.

TILAB Telecom Italia Lab.

UAV Unmanned Aerial Vehicle.

VANT Veículo Aéreo Não-Tripulado.

VHDL Very High Speed Integrated Circuit Hardware Description Language.

SUMÁRIO

1 INTRODUÇÃO

1.1 Introdução

SMA (Sistema Multi-Agente) é uma tecnologia orientada a aplicações com requisitos de flexibilidade, robustez e reconfigurabilidade (PEREIRA; RODRIGUES; LEITAO, 2012). Devido às suas características, existe um interesse de aplicação e pesquisa sobre SMAs em vários campos de atuação, como sistemas embarcados (BOSSE, 2014), prédios inteligentes (ROSCIA; LONGO; LAZAROIU, 2013), segurança (MEGHERBI; KIM; MADERA, 2013), missões com VANTs (Veículos Aéreos Não-Tripulados) (URE et al., 2014), monitoramento de desastres naturais (MUSTAPHA; MCHEICK; MELLOULI, 2013), redes de transmissão inteligentes (VRBA et al., 2014), aplicações industriais (LEITAO; MARIK; VRBA, 2013) e redes adaptativas em geral (SAYED, 2014).

Um SMA é composto por *Agentes*. Uma definição, de acordo com (WOOLDRIDGE, 2002), é de que um Agente é um sistema computacional capaz de agir de forma independente ou autônoma, por conta de seu usuário ou dono, descobrindo o que necessita ser feito para atingir objetivos, ao invés de receber as instruções diretamente.

Um SMA é um sistema composto por um número de Agentes, que interagem uns com os outros. No caso mais genérico, Agentes tomam decisões por conta de seus donos ou usuários que possuem diferentes objetivos ou motivações. Para que consigam interagir, Agentes necessitam da habilidade de cooperar, coordenar e negociar uns com os outros, de forma semelhante como fazem os humanos.

Em (RUSSELL; NORVIG, 2003) um Agente inteligente é definido como uma entidade que percebe o seu ambiente através de sensores e age sobre o ambiente através de atuadores.

Aparentemente não existe uma definição universal para Agentes. No entanto, existe consenso de que a autonomia e a habilidade de interagir com outros sistemas são as principais habilidades de Agentes. Para além desses atributos, diferentes capacidades recebem mais relevância dependendo da sua aplicação (RUDOWSKY, 2004).

Agentes podem estar inseridos em diversos ambientes. Exemplos de ambientes são ambientes reais, como o mundo físico ao nosso redor, ou ambientes virtuais como ambientes em software de um SO (Sistema Operacional) (WOOLDRIDGE, 2002).

Uma das áreas onde há interesse de aplicação de Agentes é no ambiente industrial. Em (PEREIRA; RODRIGUES; LEITAO, 2012) é apontado que SMAs são uma

tecnologia adequada para este fim pois é capaz de implementar sistemas que exigem flexibilidade, robustez e reconfigurabilidade. Apesar da adequação dos princípios de SMAs para resolver as necessidades da indústria, a verdadeira implantação de SMAs para aplicações industriais ainda não pode ser ser considerada resolvida. Um dos motivos apontados é a necessidade de se integrar aos SMAs os equipamentos de manufatura já existentes.

De acordo com (LEITAO; MARIK; VRBA, 2013), o uso de Agentes na indústria possibilita muitas vantagens, especificamente em termos de robustez, escalabilidade, reconfigurabilidade e produtividade. No entanto, o estudo indica que existem ainda problemas que impedem a utilização em larga escala de conceitos de Agentes nesta área, como questionável retorno do investimento, falta de ferramentas de desenvolvimento e padrões, falta de projetos existentes e pessoas especializadas.

Existem padrões e frameworks para o desenvolvimento de SMAs, que podem ser utilizados para reduzir os problemas apresentados. Um exemplo de padrão é dado pela FIPA (Foundation for Intelligent Physical Agents), na forma de um padrão para interação de Agentes inteligentes (O'BRIEN; NICOL, 1998). Este padrão é implementado por diversos frameworks, como o JADE (Java Agent Development Framework) (BELLIFEMINE; POGGI; RIMASSA, 2001) (TILAB, 2014). Através do uso destes padrões, podem ser desenvolvidos componentes capazes de comunicação com outros frameworks compatíveis, e portanto reutilizáveis.

Algumas das outras vantagens do JADE são permitir que Agentes sejam desenvolvidos numa linguagem de alto nível (Java), prover funcionalidades para facilitar a gerência e depuração do sistema, e possibilitar a migração de Agentes entre plataformas compatíveis.

Outro obstáculo apontado como um desafio para a aplicação de Agentes na indústria é a necessidade de controle em tempo-real (PEREIRA; CARRO, 2007). Em (KROL; NOWAKOWSKI, 2013) e (FILGUEIRAS; LUNG; RECH, 2012) são feitos estudos mais aprofundados a respeito dos problemas para adoção de Agentes em larga escala.

Outro tópico emergente no desenvolvimento de Agentes é a implementação de Agentes em hardware, além da comunicação entre Agentes em hardware e Agentes em software. Alguns trabalhos discutem implementações de SMAs com Agentes em hardware, como (BELKACEMI et al., 2011), (SCHNEIDER; NAGGATZ; SPALLEK, 2007) e (CEMIN; GOTZ; PEREIRA, 2012). A maior parte dos trabalhos observado nesta área resulta em propostas de implementação que permitem a comunicação entre Agentes da mesma natureza de implementação (Agentes em hardware comunicando com Agentes em

hardware, ou Agentes em software comunicando com Agentes em software) ou permitem a comunicação entre Agentes de naturezas diferentes através de um protocolo proprietário. Os tópicos de comunicação entre naturezas de implementação e migração de Agentes de software para hardware e vice-versa também é pouco abordado.

O trabalho descrito em (CEMIN; GOTZ; PEREIRA, 2012) detalha uma proposta para a implementação de Agentes em hardware e permite comunicação e migração entre Agentes de hardware e software. Neste trabalho, Agentes em hardware são compostos de um FPGA (Field-programmable Gate Array) que implementa a funcionalidade de hardware do Agente, e um outro processador acoplado provê uma interface que permite a transmissão de mensagens JADE para execução em hardware. Um *middleware* adicional é proposto para gerenciar a configuração do FPGA no momento da migração do Agente. O *middleware* tem um papel importante neste trabalho, pois ele cria uma abstração para o desenvolvimento dos Agentes, simplifica a implementação dos Agentes, e abstrai as requisições aos Agentes implementados em hardware. Através do uso do JADE para as duas naturezas de implementação, é possível que os Agentes em hardware se comuniquem com Agentes em software através da interface padronizada e ao mesmo tempo tenham sua implementação otimizada.

Algumas limitações discutidas em (CEMIN; GOTZ; PEREIRA, 2012) são que a reconfiguração completa do FPGA pode ser cara em tempo de execução e recursos, e a necessidade de prover um processador adicional apenas para fazer a interface entre o FPGA e o JADE. Da forma como foi apresentada, a arquitetura proposta também provê a limitação de que apenas um Agente pode existir em um FPGA por vez. A ideia de reconfiguração parcial do FPGA também é citada como uma alternativa para alguns destes problemas, porém é deixada como trabalho futuro. A expectativa é que a adição de reconfiguração parcial de FPGA a este trabalho permita tempos reduzidos de reconfiguração e o aumento da flexibilidade, possibilitando que vários Agentes em hardware coexistam em apenas um FPGA.

Um incremento adicional que pode ser feito ao estudo de (CEMIN; GOTZ; PEREIRA, 2012) é o uso de métodos de comunicação de alto desempenho entre o processador e a função do Agente no FPGA. O middleware original depende do uso de acessos a registradores no estilo MMIO (Memory Mapped Input/Output), o que se torna sub-ótimo em aplicações que requerem transferência de maiores quantidades de dados, como processamento de imagens de alta resolução ou vídeo.. O uso de um processador externo é um dos fatores limitantes para a comunicação entre o processador e o FPGA. Uma solu-

ção alternativa seria explorar soluções que acoplam um processador ao mesmo chip do FPGA. Isto permite o uso de ferramentas e barramentos mais integrados dentro do FPGA, o que pode ser vantajoso para a arquitetura proposta. O uso de barramentos mais integrados pode favorecer o uso de DMA (Direct Memory Access), por exemplo, podendo adicionar capacidade de processamento de maiores quantidades de dados em hardware.

Este trabalho pode ser visto como uma extensão a (CEMIN; GOTZ; PEREIRA, 2012), e propõe alterações no framework no âmbito de métodos de comunicação entre processador e função em hardware, forma de implementação do processador e melhorias na reconfiguração do FPGA.

As modificações propostas ao framework são avaliadas em um conjunto de aplicações de processamento de sinais e imagens, incluindo um estudo de caso de uma aplicação para agricultura de precisão utilizando processamento de imagem utilizando VANTs. Neste estudo de caso, imagens aéreas obtidas através de um VANT são processadas por um algoritmo de segmentação que executa a segmentação de pixels de solo e plantações. São avaliados o tempo de execução em diferentes plataformas, para verificar o impacto da utilização do framework na execução do algoritmo.

1.2 Objetivos

A principal contribuição deste trabalho é plataforma para SMAs envolvendo hardware reconfigurável. Isto inclui a proposta para a arquitetura utilizada, e a implementação de referência realizada. Estes tópicos são descritos em detalhe no Capítulo 4. Seguem algumas dos objetivos considerados:

- Dar continuidade ao trabalho relacionado anterior, apresentado em (CEMIN; GOTZ; PEREIRA, 2012).
- Adicionar a funcionalidade de reconfiguração parcial, estudar maneiras de como esta funcionalidade pode ser explorada, avaliar resultados práticos através de uma implementação. Medir resultados através da implementação.
- Utilizar um processador integrado ao FPGA no mesmo chip, para que seja possível explorar formas mais complexas de comunicação entre o processador e o FPGA.
- Aplicar o trabalho e implementar estudos de caso mais complexos, de Agentes realizando processamento que demanda mais recursos de processamento, como o uso de processamento digital de imagens, e o uso de Agentes em VANTs. Realizar

medidas de desempenho para os estudos de caso apresentados.

Este trabalho não tem como um objetivo primário propor um SMA para resolver um problema em específico, mas sim propor a plataforma na qual aplicações de Agentes podem ser implementadas tomando vantagem de plataformas reconfiguráveis. Os estudos de caso propostos têm por objetivo avaliar a plataforma e mostrar aplicações com as quais a plataforma pode ser utilizada.

1.3 Organização

Este trabalho é organizado da seguinte forma: Capítulo 2 define os conceitos técnicos importantes para entendimento do trabalho e da solução proposta. Capítulo 3 faz uma revisão do estado da arte de SMAs na automação. Neste capítulo também são apresentados as oportunidades de contribuição para o estado da arte, baseado nos artigos revisados. Capítulo 4 apresenta a plataforma proposta e discute a arquitetura e implementação de referência em detalhe. Capítulo 5 discute a aplicação do framework em estudos de caso, incluindo o estudo de caso de agricultura de precisão com VANTs. Neste capítulo ainda são apresentados os resultados experimentais para avaliar as opções de implementação. Capítulo 6 apresenta conclusões e discute possibilidades de trabalho futuro.

2 CONCEITOS BÁSICOS

Neste capítulo, serão apresentados definição e conceitos básicos sobre SMAs, que serão as definições consideradas no decorrer do trabalho.

Na Seção 2.1 é discutida uma definição de Agente e algumas aplicações. É apresentada também uma introdução ao framework JADE, utilizado no trabalho para a implementação de SMAs, suas características e funcionalidades.

Na Seção 2.3 são discutidas arquiteturas reconfiguráveis e alguns conceitos de hardware relacionados ao trabalho.

Na Seção 2.4 são discutidos conceitos de software relacionados ao trabalho, principalmente relacionados a Linux embarcado.

2.1 Sistemas Multi-Agente

A definição do termo Agente diverge entre diferentes autores e existe discussão e debate relacionado a este assunto. Atualmente, não existe uma definição aceita universalmente para o termo. Existe o consenso geral de que "autonomia" é a ideia central para a definição de Agente, porém existe pouca concordância entre autores com relação ao restante da definição.

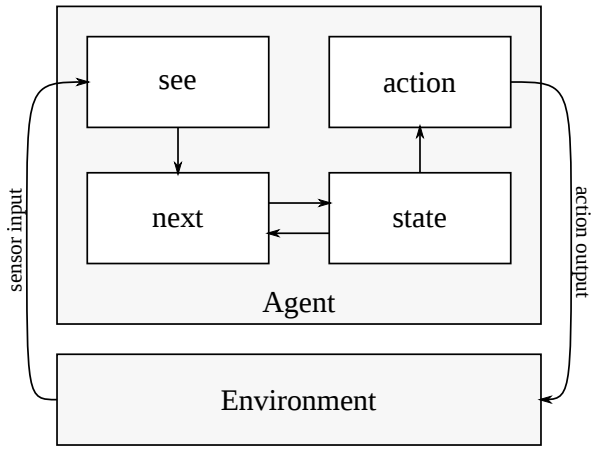
Parte da dificuldade existe pois vários dos atributos associados com Agentes são importantes para diferentes domínios. Desta forma, para algumas aplicações, a habilidade dos Agentes em aprender através de suas experiências é de fundamental importância; para outras aplicações, aprender não é importante e nem desejável. Mesmo assim, é importante existir uma definição base; caso contrário o termo pode acabar perdendo seu significado. A definição apresentada aqui é adaptada de (WOOLDRIDGE, 2002):

Um Agente é um sistema computacional, situado em um dado ambiente, e que é capaz de realizar ações autônomas, visando atingir os seus objetivos de projeto.

SMAs são sistemas compostos por múltiplos Agentes capazes de interagir entre si. Agentes têm duas importantes capacidades: Primeiro, eles são capazes de realizar ações de forma autônoma, ou decidir por si próprios o que precisam fazer para satisfazer os seus objetivos de projeto. Segundo, eles são capazes de interagir com outros Agentes, não simplesmente como troca de dados, mas capazes de participar em ações análogas a atividades sociais como: cooperação, coordenação ou negociação.

A Figura 2.1 ilustra a interação de um Agente com seu ambiente.

Figura 2.1 – Ambiente onde um Agente está inserido, com suas medidas, ações e percepções



Fonte: Autor, com base em (WOOLDRIDGE, 2002)

2.2 JADE

O JADE é um *middleware* desenvolvido inicialmente por TILAB (Telecom Italia Lab) para o desenvolvimento de SMAs distribuídos para aplicações baseadas em comunicação ponto-a-ponto. O JADE é disponibilizado como um projeto de software livre, sob licença LGPL (Lesser General Public License), sendo possível acessar, modificar, contribuir e distribuir o seu código fonte. O JADE é desenvolvido em Java, portanto é necessário que o sistema alvo tenha à disposição uma JVM (Java Virtual Machine). Na versão 4.4.0, o JADE requer uma JVM com suporte à versão 1.5 do Java ou maior.

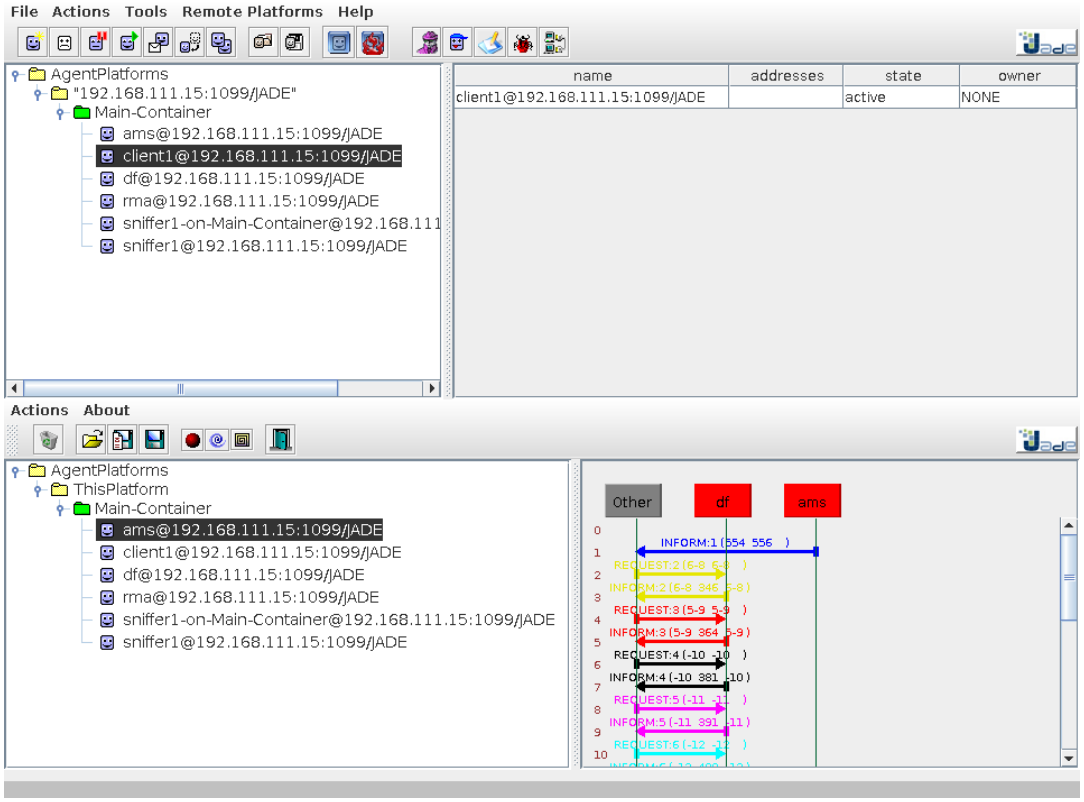
O JADE é baseado nos seguintes princípios:

- Interoperabilidade: o JADE é complacente com as especificações da FIPA. Como uma consequência, os Agentes implementados com o JADE são capazes de interoperar com Agentes desenvolvidos com outros frameworks que implementam o mesmo padrão.
- Uniformidade e portabilidade: o JADE provê um conjunto homogêneo de chamadas de API (Application Programming Interface) que são independentes da rede ou de versões do Java. O ambiente do JADE provê a mesma API para os ambientes J2EE, J2SE e J2ME.
- Facilidade de uso: a complexidade do *middleware* é escondida sob uma API projetada para ser intuitiva e simples.

- Filosofia *pay-as-you-go*: Funcionalidades que não são necessárias para uma aplicação não causam *overhead* computacional e não requerem ação do programador.

O JADE provê ferramentas que auxiliam na depuração e desenvolvimentos de Agentes, como uma interface gráfica implementada pelo Agente RMA (Remote Management Agent). A interface gráfica do Agente RMA é mostrada na Figura 2.2. Através desta interface, é possível observar os Agentes presentes no container, analisar as mensagens trocadas entre os Agentes, e tomar ações como disparar a migração de Agentes entre containers.

Figura 2.2 – Captura de tela de GUI do JADE. Interfaces providas pelos Agentes RMA e Sniffer



Fonte: Autor

Além do RMA, outro exemplo de Agente útil para o desenvolvimento é o Agente *sniffer*, capaz de interceptar mensagens durante a execução e demonstrá-las graficamente. A janela do Agente *sniffer* também é mostrado na Figura 2.2. Na parte inferior da figura, é possível ver o uso do *sniffer* mostrando a troca de mensagens entre dois Agentes selecionados e mensagens enviadas para Agentes não selecionados.

Além da abstração para implementação de Agentes, o JADE provê um modelo de comunicação de mensagens assíncronas ponto-a-ponto, um serviço de *páginas amarelas*

no padrão *publish-subscribe* para descoberta de outros serviços disponíveis no SMA, além de outras funcionalidades avançadas para o desenvolvimento de sistemas distribuídos.

O modelo de comunicação do JADE é baseado em troca de mensagens assíncronas seguindo a linguagem ACL (Agent Communication Language) especificada pela FIPA. Isso permite que o JADE se comunique com outros frameworks que utilizem esta mesma linguagem.

A ACL é uma linguagem de comunicação que define tipos de mensagem que permitem que os Agentes troquem mensagens informando intenções como: informar, solicitar, perguntar, entre outros (O'BRIEN; NICOL, 1998). Para isso, as mensagens incluem campos designados para interações entre Agentes, como as apresentadas na Tabela 2.1.

É possível observar os Agentes trocando mensagens e identificar seus respectivos Atos Comunicativos na Figura 2.2.

O JADE provê alguns Agentes especiais para controlar o seu funcionamento, como os Agentes DF (Directory Facilitator) e AMS (Agent Management System). O Agente DF provê o diretório de *páginas amarelas* que anuncia os serviços presentes no SMA. Através do Agente DF, os Agentes podem registrar serviços que são capazes de oferecer. Assim, quando outros Agentes precisarem solicitar um serviço, podem consultar o DF para descobrir quais Agentes podem oferecer aquele serviço. Após descobrir os serviços disponíveis pelo DF, um Agente pode contatar os outros em comunicação ponto-a-ponto e enviar uma proposta de realização de ação, através de mensagens com os Atos Comunicativos adequados. O JADE permite ainda a definição de Ontologias, que descrevem quais mensagens e Atos Comunicativos devem ser utilizados para que os Agentes possam de forma que as mensagens façam sentido.

No JADE, os Agentes existem no topo de *plataformas*, que lhes provê serviços básicos, como troca de mensagens. Uma *plataforma* é composta por um ou mais containers. Esta relação é representada na Figura 2.3. Os containers podem estar localizados em nodos diferentes mas estarem registrados em uma mesma *plataforma*, criando assim uma *plataforma* distribuída. Cada container pode conter zero ou mais Agentes. O Agente AMS é o Agente que controla a plataforma, sendo o único que pode criar ou destruir outros Agentes, criar ou destruir containers ou finalizar a *plataforma*.

Por fim, o JADE possibilita a realização de migração ou clonagem de Agentes entre containers, ou nodos do SMA, dentro de uma mesma *plataforma*. Para que seja possível realizar estas operações, os Agentes devem implementar as funcionalidades correspondentes providas pelo framework. Para realizar as ações de migração ou clonagem,

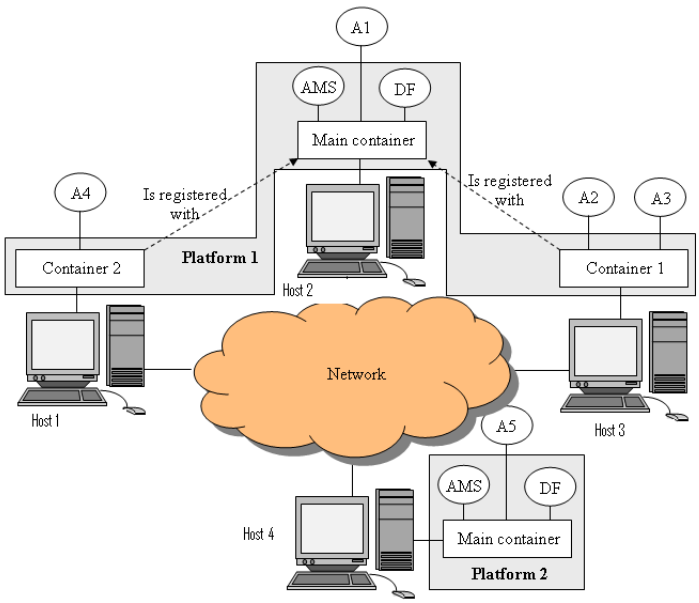
Tabela 2.1 – Atos Comunicativos na FIPA-ACL

Ato Comunicativo	Significado
ACCEPT-PROPOSAL	Aceitar uma requisição proposta para realizar uma ação.
AGREE	Aceitar realizar uma ação, possivelmente no futuro.
CANCEL	Cancelar uma ação previamente requisitada.
CFP	Buscar por propostas para realizar uma dada ação.
CONFIRM	Confirmar que uma determinada proposição é verdadera.
DISCONFIRM	Desconfirmar que uma determinada proposição é verdadeira.
FAILURE	Informar que houve uma tentativa de realizar uma ação, porém houve uma falha.
INFORM	Informar que uma determinada proposição é verdadeira.
NOT-UNDERSTOOD	Informar que não foi possivel entender uma determinada requisião.
PROPOSE	Enviar uma proposta para realizar uma ação.
QUERY-IF	Perguntar a um Agente se uma determinada proposição é verdadeira.
QUERY-REF	Pedir a um Agente um objeto referenciado por uma expressão.
REFUSE	Recusar a realização de uma ação, informando o motivo.
REJECT-PROPOSAL	Recusar a realização de uma ação, durante uma negociação.
REQUEST	Solicitar a realização de uma ação.
REQUEST-WHEN	Solicitar a realização de uma ação quando uma determinada proposição se tornar verdadeira.
REQUEST-WHENEVER	Solicitar a realização de uma ação sempre que uma determinada proposição for verdadeira.
SUBSCRIBE	Solicitar ser informado sempre que houver uma mudança acontecer em um determinado objeto.

Fonte: (JUNEJA; JAGGA; SINGH, 2015)

os Agentes devem enviar solicitações ao AMS. O JADE implementa a comunicação com o AMS para estes fins como parte do framework, facilitando a realização destas operações.

Figura 2.3 – Arquitetura de containers e plataformas do JADE



Fonte: (CAIRE., 2009)

2.3 Arquiteturas Reconfiguráveis

Arquiteturas Reconfiguráveis se refere a arquiteturas que combinam a flexibilidade de software com o alto desempenho de hardware, através de circuitos reconfiguráveis como os FPGAs. A principal diferença destes dispositivos com relação a outros circuitos integrados como os ASICs (Application-specific integrated circuits) é a capacidade de "carregar" um novo circuito no dispositivo de forma dinâmica. Existem outros tipos de reconfiguráveis, como os CPLDs (Complex programmable logic devices), que apresentam arquiteturas e características diferenciadas. Por exemplo, os CPLDs se diferenciam dos FPGAs por serem mais simples, oferecendo majoritariamente um arranjo de circuitos combinacionais com poucos registradores, o que os torna mais simples de projetar. Por outro lado, são mais limitados.

Os dispositivos FPGA possuem maior complexidade de projeto porém são de maior interesse para este trabalho devido às suas capacidades avançadas, em especial as possibilidades de reconfiguração dinâmica.

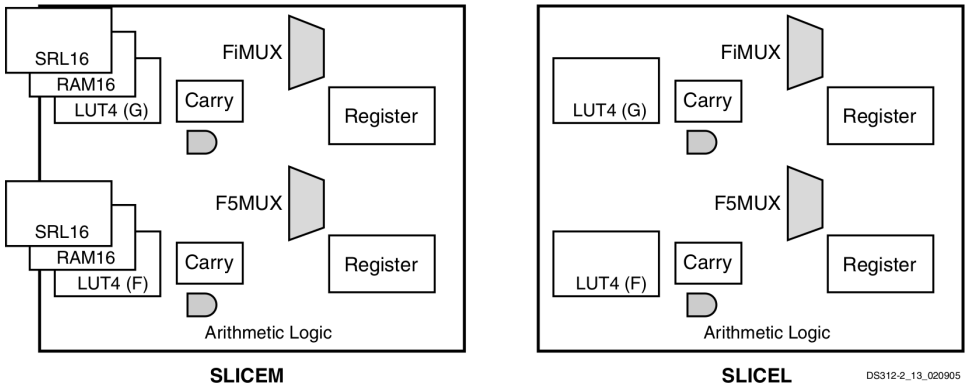
2.3.1 FPGA

Um dispositivo FPGA é um circuito integrado projetado para ser programável após sua fabricação para atender uma aplicação específica. A configuração de dispositivos FPGA é geralmente especificada utilizando uma HDL (Hardware Description Language), de forma similar à forma como é feita para um ASIC.

Um FPGA contém uma matriz de blocos lógicos reconfiguráveis (matriz reconfigurável), e uma hierarquia de interconexões reconfiguráveis que possibilita que os blocos sejam conectados logicamente após sua fabricação. Cada bloco lógico pode ser configurado para desempenhar funções como portas lógicas AND, XOR, ou desempenhar funções combinatoriais mais complexas.

Na maioria dos FPGAs, os blocos lógicos também incluem elementos de memória, que podem ir desde simples *flip-flops* ou blocos mais complexos de memória como as BRAMs (Block Rams). Os FPGAs também podem incluir blocos especializados para operações matemáticas, como multiplicadores, unidades de ponto flutuante, ou até mesmo processadores completos, conforme discutido na Seção 2.3.2. Um exemplo de FPGA com recursos como este é o Arria 10 (Intel Corporation, 2018). A Figura 2.4, obtida em (XILINX, 2013), mostra um exemplo de conteúdo de blocos lógicos na família Spartan 3E da Xilinx, denominados *slices*.

Figura 2.4 – Recursos de um *slice* programável em um FPGA Xilinx Spartan-3E



Fonte: (XILINX, 2013)

O uso de FPGAs permite realizar computação reconfigurável, que combina alguns dos benefícios da maior flexibilidade do uso de software com o alto desempenho das arquiteturas reconfiguráveis. A principal diferença do uso desta arquitetura quando comparado com o uso de microprocessadores é a habilidade de se fazer mudanças subs-

tanciais no caminho dos dados em hardware, tanto no fluxo de dados quando no caminho de controle (YANG et al., 2014).

Quando comparado ao uso de um ASIC customizado, a vantagem de se utilizar FPGAs é a possibilidade de se alterar o hardware em tempo de execução através da carga de um circuito diferente na matriz reconfigurável.

Para criar o circuito a ser carregado em um FPGA, um projetista tradicionalmente descreve o circuito utilizando um editor de esquemáticos, ou mais comumente utilizando uma descrição HDL, como Verilog ou VHDL (Very High Speed Integrated Circuit Hardware Description Language). Existem também o desenvolvimento de ferramentas para descrição de circuitos em mais alto nível, como as ferramentas MATLAB® ou Xilinx Vivado, que oferecem a opção de projeto HLS (High-level synthesis). As ferramentas HLS extraem a descrição do circuito a partir de uma descrição funcional ou de diagramas de blocos. Em qualquer um dos formatos de entrada, a descrição é passada a uma sequência de ferramentas de EDA (Electronic Design Automation), que realizam a síntese do circuito e por fim a geração de um *bitstream*. O *bitstream* é o arquivo a sequência de bits que deve ser carregada à memória de configuração de um FPGA para que o mesmo passe a desempenhar a função do circuito desejado. Em geral, a menos que esteja sendo utilizado reconfiguração parcial do FPGA (descrito na Seção 2.3.5), o *bitstream* contém a configuração de cada componente do FPGA (blocos lógicos, interconexões) para a implementação do circuito. As ferramentas EDA e os formatos de *bitstream* são em geral proprietários dos fornecedores de dispositivos FPGA.

Os dispositivos FPGA em geral são implementados em memória volátil, de forma que devem ser reprogramados a cada vez que são alimentados. Os FPGAs podem ser configurados de diversas formas, entre elas: a partir de portas de teste JTAG (Joint Test Action Group), a partir de interfaces seriais ou paralelas conectadas a um processador externo, a partir de interfaces específicas de configuração (como o Xilinx HWICAP (Hardware Internal Configuration Access Port)). Alguns dispositivos FPGA podem ser configurados para se autoconfigurar a partir de uma memória externa não-volátil. Outros dispositivos semelhantes como os CPLD são implementados com memória não-volátil e portanto não precisam ser reconfigurados a cada alimentação, porém em geral possuem menos recursos e custo mais elevado em relação aos recursos que oferecem.

2.3.2 Processadores embarcados em FPGA

É comum em sistemas embarcados a utilização de um processador embarcado ou microcontrolador em conjunto com um FPGA. Neste caso, uma parte do sistema pode ser executada em software no processador com maior flexibilidade e este se comunica com lógica programável no FPGA para partes da aplicação que possuem requisitos melhores atendidos em hardware. Uma implementação deste tipo pode ser realizada com o uso de um SOC (System on Chip) contendo o processador e um FPGA em outro chip. Uma vantagem desta implementação é a possibilidade do uso de componentes de prateleira. No entanto, uma solução neste formato com múltiplos chips pode ser pouco eficiente em alguns casos. Exemplos são casos onde é necessária comunicação de alto desempenho entre o software e a lógica programável, ou se as dimensões físicas do sistema embarcado requerem a implementação com menor área.

Uma forma alternativa de solucionar este problema seria, após validação da lógica programável, a fabricação de um ASIC customizado contendo o processador e a lógica programável. No entanto, projetos com ASIC possuem custos muito mais elevados, além de outros riscos de projeto que tornam isso não praticável para projetos com baixo ou médio volume.

Em sistemas embarcados, existem duas soluções que permitem o uso de um processador de forma integrada ao mesmo chip da lógica programável. Uma forma é sintetizar um processador utilizando lógica programável, e outra é utilizar chips que possuem um processador embarcado embutido em conjunto com o FPGA, nos denominados SOC FPGA. Estas formas são discutidas na Seção 2.3.3 e Seção 2.3.4.

2.3.3 Processadores soft core

Uma solução para se obter uma interface melhor entre processador e FPGA em sistemas embarcados é realizar a síntese de um processador utilizando lógica programável, sendo assim necessário utilizar somente um único chip para o FPGA. Estes processadores, sintetizados em lógica programável, são denominados também processadores soft core. Os processadores soft core apresentam a vantagem de serem altamente customizáveis, podendo ser reduzidos para consumir mínima área de FPGA e operar com mínima frequência, de forma semelhante a um microcontrolador, ou com conjuntos de recursos completos como FPU (Floating-point unit) e MMU (Memory Management Unit), que

permitem execução de sistemas operacionais como o Linux.

Mesmo assim, esta solução não atende todas as aplicações, pois mesmo com o suporte ao máximo número de otimizações possíveis, os processadores soft core possuem desempenho limitado e operam a frequências limitadas, que podem não ser suficientes para aplicações que demandam muito poder de CPU (Central Processing Unit). Além disso, os processadores soft core podem ocupar uma parte significativa da área reconfigurável do FPGA. Exemplos de processadores soft core são o Nios2, da Altera, e o MicroBlaze, da Xilinx.

Em (MATTSSON; CHRISTENSSON, 2004), é feita uma revisão e comparação de algumas opções de processadores soft core existentes.

2.3.4 SOC FPGA

Uma solução que vem se tornando mais acessível são FPGAs que incluem um processador embarcado no mesmo chip, nos chamados SOC FPGA (ALTERA CORPORATION, 2013). Os processadores presentes nestes chips apresentam desempenho equivalente ao de um processador embarcado convencional, e possuem barramentos conectados à área reconfigurável do FPGA, oferecendo bom desempenho para comunicação com o FPGA. Por outro lado, oferecem menos flexibilidade do que os soft core pois as conexões do processador com o FPGA são pré-definidas. Além disso, dependendo da aplicação, os processadores nestes FPGAs podem acabar sendo subaproveitados. Exemplos de SOC FPGA são a linha Cyclone V, da Altera, e a linha Zynq, da Xilinx.

2.3.5 Reconfiguração Parcial

Assim como projeto de software, hardware em dispositivos programáveis pode ser projetados como blocos, e em algumas aplicações pode ser vantajoso substituí-los durante a execução.

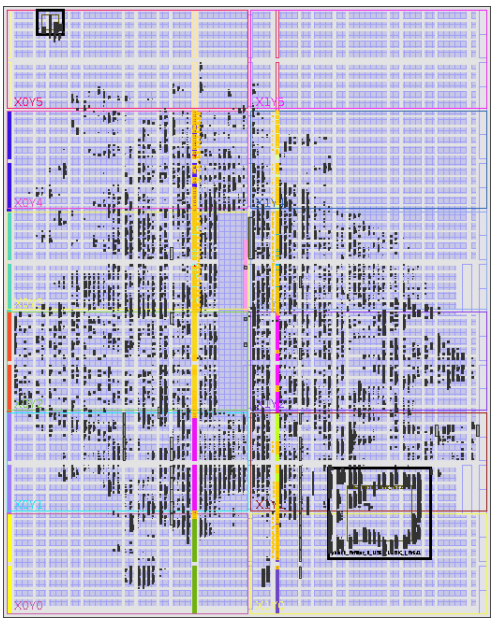
Reconfiguração parcial é o processo de reconfiguração de uma parte do hardware implementado em um dispositivo programável enquanto o restante permanece em funcionamento. Esta funcionalidade é suportada na maioria dos FPGAs atuais.

Através desta funcionalidade, é possível implementar hardware adaptável que oferece aceleração otimizada para a atividade que precisa desempenhar no momento. Como

exemplo, um dispositivo de comunicação pode carregar diferentes módulos de criptografia em hardware e automaticamente reconfigurar o seu bloco de criptografia dependendo da comunicação que precisa realizar. Caso o projeto no FPGA inclua um processador, o processador pode alterar um periférico acelerador em hardware sem a necessidade de reiniciar o FPGA.

Para se utilizar reconfiguração parcial, o FPGA é dividido entre a região *estática* e são delimitadas regiões *reconfiguráveis*. A Figura 2.5 mostra uma divisão entre essas regiões, realizada na ferramenta Xilinx PlanAhead.

Figura 2.5 – Projeto implementado com o PlanAhead no FPGA Virtex-6 XC6VLX240T – os retângulos marcados a noroeste e sudeste são as áreas reservadas para reconfiguração parcial



Fonte: Autor

Para se utilizar reconfiguração parcial, são necessários *bitstreams* adicionais para as regiões reconfiguráveis. Existem dois estilos de implementação de reconfiguração parcial em dispositivos FPGA: baseado em módulos ou baseado em diferenças.

Reconfiguração parcial baseada em módulos permite a reconfiguração de partes modulares do projeto. Neste estilo de projeto, as regiões reconfiguráveis são capazes de implementar uma interface de pinos padrão, que pode aceitar qualquer bloco reconfigurável compatível. Desta forma, para cada módulo reconfigurável no projeto, um *bitstream* parcial é necessário.

A reconfiguração parcial baseada em diferenças permite a reconfiguração de partes modulares do projeto, porém neste estilo de projeto, o *bitstream* parcial contém a infor-

mação de localização absoluta no FPGA. Desta forma, o número de *bitstreams* parciais necessário é equivalente ao o número de blocos reconfiguráveis multiplicado pelo número de opções de reconfiguração desejados.

É possível iniciar a reconfiguração parcial do FPGAs de diversas formas, semelhante à configuração completa do FPGA (discutido na Seção 2.3.1). Em particular, é possível realizar a reconfiguração parcial de um FPGA a partir de um processador que esteja executando dentro do próprio FPGA, como a partir dos processadores discutidos na Seção 2.3.2. Isto é possível pois não é necessário reiniciar o FPGA para realizar a reconfiguração parcial. Nos FPGAs da Xilinx, isto pode ser realizado através do bloco de hardware denominado HWICAP. O HWICAP é discutido em mais detalhes na Seção 4.3.3. Um estudo de caso de reconfiguração parcial utilizando um processador soft core MicroBlaze e Linux embarcado é discutido em (WILLIAMS; BERGMANN, 2004).

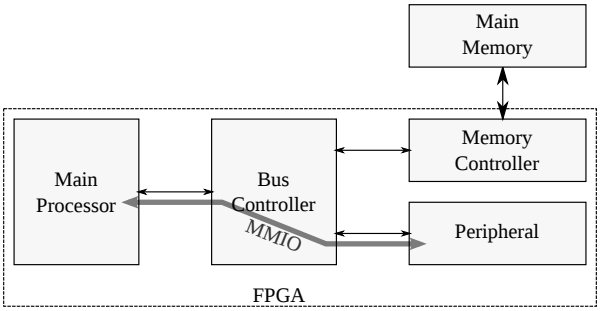
2.3.6 Métodos de comunicação entre o processador e o hardware

Na Figura 2.6, são mostrados os tipos de acesso à memória que podem ocorrer para acesso a um periférico de hardware mapeado em memória.

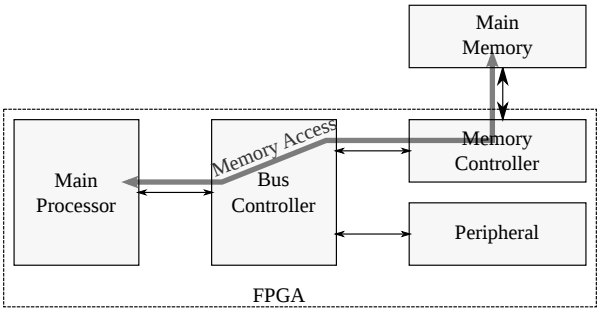
A forma mais simples de se acessar o periférico é através de registradores mapeados em memória, em acessos MMIO, ou também conhecidos como acessos PIO (Programmed Input/Output). Este acesso é representado na Figura 2.7(a). Em acessos MMIO, os registradores de configuração e estado do periférico são acessíveis no mesmo espaço de endereçamento da memória principal. Ao acessar estes endereços dedicados ao periférico, é possível transferir dados entre o periférico e o processador. Este tipo de acesso tem a vantagem de ser o mais simples e, dependendo do dispositivo, pode ser suficiente para interfacear com o dispositivo. No entanto, estes métodos requerem que o processador transmita os dados em pequenos blocos como 1 ou 4 bytes, demandando o uso do processador durante toda a transferência dos dados. Por utilizar transferências individuais ao invés de transferências em rajada, fazem uso menos eficiente do barramento. Por isso, são uma forma sub ótima de transmitir maiores volumes de dados, como imagens. Acessos MMIO são adequados para que o processador realize controle e configuração do dispositivo, ou requisição de o início de operações de processamento. Acessos MMIO podem ser utilizados também para a leitura de status e informações dos periféricos.

Dispositivos que requerem maior desempenho devido à transferência de maiores quantidades de dados podem fazer o uso da técnica de DMA. Neste caso, o dispositivo

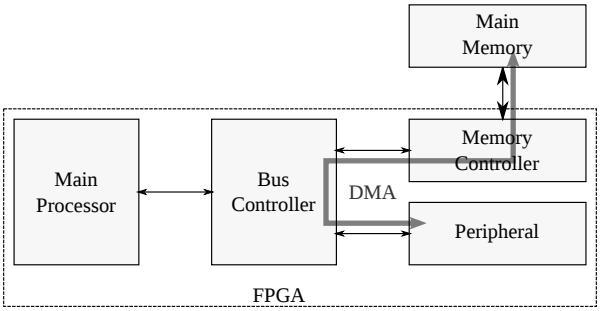
Figura 2.6 – Representação de acessos à memória, MMIO e DMA



(a) Acesso a registradores de um periférico por MMIO



(b) Acesso normal do processador à memória



(c) Acesso DMA de a um periférico à memória

Fonte: Autor

deve estar inserido em um barramento que suporte acessos DMA e possuir recursos em hardware que o permitem iniciar transações de acesso à memória. Em um cenário onde o software detém os dados a serem processados no periférico, o software primeiro armazena os dados em um buffer realizando acessos normais à memória, conforme representado na Figura 2.7(b). Após isso, o software instrui o periférico a acessar a memória para obter os dados a processar. O software ainda realiza acessos MMIO para acessar o periférico, mas apenas registrar no periférico o endereço de memória dos dados, e para iniciar a operação de DMA, Quando instruído a realizar acessos DMA, o periférico realiza acessos conforme representado na Figura 2.7(c). É comum neste cenário que o periférico possua um sinal

de interrupção para sinalizar ao processador quando o processamento foi finalizado.

O uso de DMA neste cenário tem diversas vantagens: o processador não fica bloqueado transmitindo dados ao periférico, o periférico consegue receber os dados mais rapidamente realizando acessos em rajada à memória, e o processador fica livre para realizar outras operações, sendo interrompido apenas quando os dados relevantes estão prontos.

2.4 Linux em sistemas embarcados

GNU/Linux, ou somente Linux, é um termo comumente utilizado para se referir a SOs baseados no kernel (ou núcleo) do SO Linux.

Um sistema Linux é composto pelo kernel, que representa uma importante parte e provê interfaceamento com o hardware, além de serviços e abstrações para execução de programas. O espaço de usuário em sistemas Linux compreende o conjunto de programas, aplicações e bibliotecas com os quais o usuário interage mais diretamente que utilizam os serviços disponibilizados pelo kernel para sua execução.

O Linux surgiu na década de 90 como um projeto de hobby e suportava inicialmente somente processadores da família 386 da Intel. Nas décadas subsequentes, devido ao modelo de desenvolvimento de software livre, o Linux se tornou um SO maduro e dominante em vários segmentos, como servidores, celulares e sistemas embarcados, além de suportar mais arquiteturas do que qualquer outro SO atualmente.

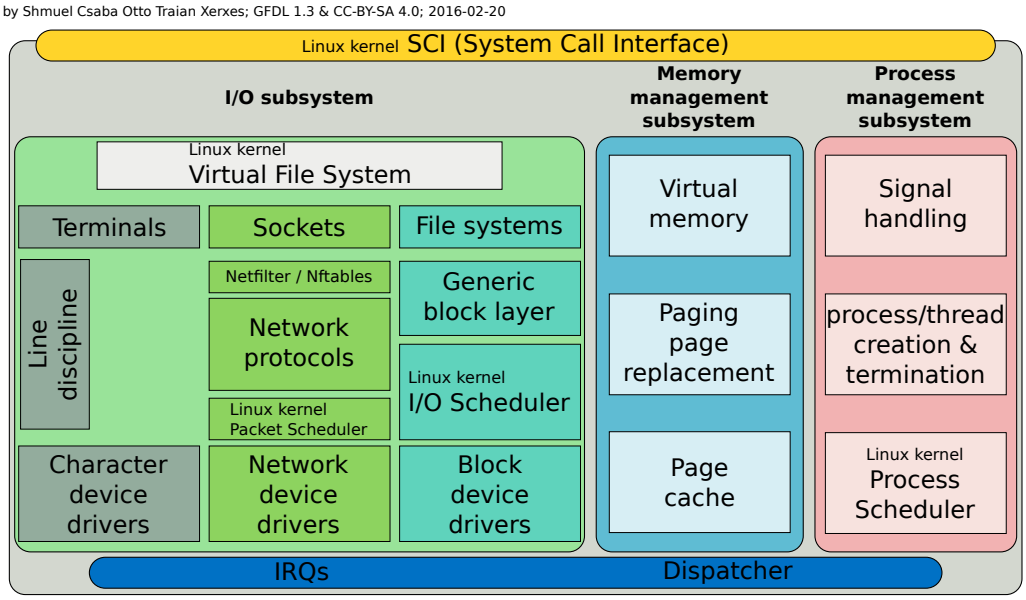
O uso do Linux é de fundamental importância neste trabalho, em especial por suas funcionalidades e compatibilidade com sistemas embarcados. Nesta seção, serão apresentados conceitos adicionais sobre Linux embarcado e sua importância no estado da arte.

2.4.1 Kernel Linux

O kernel Linux é um SO compatível com a especificação POSIX (Portable Operating System Interface) e distribuído sobre a licença GNU GPL (General Public License), sendo um dos projetos de software livre mais importantes atualmente. A Figura 2.7 apresenta uma ilustração de alto nível sobre as funcionalidades oferecidas pelo kernel.

Conforme observado na figura, Figura 2.7, o Linux provê, dentre outros, subsistema de entrada e saída (*I/O subsystem*), subsistema de gerenciamento de memória

Figura 2.7 – Ilustração simplificada da estrutura do kernel Linux



Fonte: commons.wikimedia.org (COMMONS, 2015)

(*Memory management subsystem*), e subsistema de gerenciamento de processos (*Process management subsystem*). O uso destes subsistemas em sistemas embarcados facilita o desenvolvimento das aplicações, que precisam se preocupar muito menos com detalhes de baixo nível e do hardware. Além disso, existe uma extensa gama de aplicações e bibliotecas compatíveis com qualquer sistema Linux. Como exemplo, bibliotecas de carregamento e processamento de imagem, compressão de dados, criptografia, protocolos de rede, depuração de programas, etc. Dado que o kernel provê sempre a mesma abstração para o espaço de usuário, independente da arquitetura do processador utilizado, as aplicações desenvolvidas para plataforma Linux são facilmente portáveis para outras arquiteturas.

O kernel Linux é altamente customizável, sendo possível editar sua configuração e incluir somente as funcionalidades necessárias para o sistema alvo. Outra funcionalidade relevante é a possibilidade de se construir partes do kernel que são incluídas como módulos carregados dinamicamente. Assim, é possível se construir drivers para dispositivos que serão carregados somente quando o dispositivo estiver em uso, ou somente enquanto o dispositivo de hardware estiver conectado ao sistema.

Por fim, o Linux oferece vantagens por ser um SO disponibilizado no modelo de software livre. Além do fato de ser disponibilizado sem custo, a comunidade de desenvolvimento do Linux oferece vantagens como o desenvolvimento ativo, que resulta em

um código que está sempre atualizado e com correções constantes de segurança. Existem outros benefícios menos tangíveis, como o espaço para criatividade e inovação que o ambiente de software livre oferece, que resulta em benefícios para o sistema (SYSTEMS, 2002).

2.4.2 Drivers Linux

No Linux, as aplicações de espaço de usuário não são capazes de acessar diretamente o hardware. Assim como em outros SOs, esta implementação é proposital para oferecer uma abstração entre os diversos dispositivos de hardware que oferecem uma dada funcionalidade, possibilitando às aplicações de alto nível que não tenham que se preocupar com detalhes de comunicação com o hardware. Além disso, o SO tem uma visão geral de todas as aplicações executando no sistema, e resolve os problemas de acesso concorrente ao hardware. Portanto, o SO deve oferecer interfaces para que as aplicações solicitem ao SO que realize uma determinada ação relacionada ao hardware. O código que realiza a implementação de uma interface para um dispositivo de hardware específico é comumente denominado um *driver*. Por exemplo, o SO deve oferecer interfaces para que as aplicações *escrevam* dados no hardware, ou *leiam* estados do hardware. Estas interfaces são denominadas chamadas de sistema.

Como estamos interessados em implementar hardware customizado neste trabalho, é necessário apresentar algumas formas de como implementar drivers como meios de comunicação entre as aplicações e o hardware customizado.

Um dos modelos oferecidos pelo Linux para implementação de drivers é utilizar a abstração de manipulação de arquivos. Nesta abstração, cada dispositivo é representado como arquivo em um sistema de arquivos especial. Este arquivo que representa o dispositivo é denominado *nodo de dispositivo* no Linux. Ao realizar operação como abrir, escrever, ou ler deste arquivo, o SO realiza operações análogas a estas operações no dispositivo. Desta forma, a implementação de um driver de dispositivo utilizando esta abstração consiste de implementar funções a serem chamadas quando a aplicação realiza as operações de arquivo sobre o nodo de dispositivo. A Listagem 2.1 apresenta um subconjunto da interface no Linux que expõe esta abstração.

Conforme sugerido na Listagem 2.1, para se implementar métodos de transmitir ou receber dados do dispositivo, o driver deve implementar e registrar as funções `write` e `read`, respectivamente. As operações `open` e `release` são as únicas obrigatórias,

Listagem 2.1 – Trecho da estrutura `file_operations` do Linux

```
1  struct file_operations {
2      struct module *owner;
3      ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
4      ssize_t (*write) (struct file *, const char __user *, size_t,
           loff_t *);
5      unsigned int (*poll) (struct file *, struct poll_table_struct *);
6      long (*ioctl) (struct file *, unsigned int, unsigned long);
7      int (*mmap) (struct file *, struct vm_area_struct *);
8      int (*open) (struct inode *, struct file *);
9      int (*flush) (struct file *, fl_owner_t id);
10     int (*release) (struct inode *, struct file *);
11     /* ... */
12 };
```

e são chamadas quando a aplicação começa e termina de fazer acessos ao dispositivo, respectivamente. Operações avançadas como `mmap` são opcionais e dependem do dispositivo. A operação `ioctl` oferece uma forma de a aplicação realizar operações de controle sobre a forma como vai acessar o dispositivo, ou para instruir o driver a realizar uma operação de controle no dispositivo.

Existem outras interfaces de comunicação entre as aplicações e os drivers, como o uso de sistemas de arquivos virtuais. O Linux oferece vários sistemas de arquivos virtuais com vários propósitos, como o `sysfs` e o `procfs`. As aplicações interagem de forma semelhante com estes sistemas de arquivos virtuais, porém ao invés de realizar as chamadas de sistema sobre um único nodo de dispositivo, realizam as operações em arquivos separados que representam ações diferentes.

Por fim, existem outras interfaces especializadas como para os subsistemas de rede e de armazenamento, que tratam de atender a requisitos variados.

2.4.3 Estado da arte em Linux Embarcado

SOs baseados no kernel Linux são utilizados em todos os tipos de sistemas embarcados, desde eletrônicos de consumo (como set-top boxes, smart TVs, sistemas de entretenimento em veículos), equipamentos de telecomunicação (como switches e roteadores wireless), máquinas de controle, automação industrial, equipamentos de navegação, entre outros.

Devido à sua versatilidade, sistemas operacionais baseados no kernel Linux podem também ser encontrados em dispositivos móveis, como smartphones, tablets, e outros

dispositivos de mídia.

O tema de Internet das Coisas, ou IOT (Internet of Things) é um conceito em ascensão e que pode representar a próxima revolução tecnológica. Com isso, existe no mercado uma popularização de plataformas de desenvolvimento open source e facilmente disponíveis que permitem rápida prototipação e desenvolvimento de soluções para a IOT. Exemplos de plataformas de desenvolvimento são os populares SBC (Single-board computer) Raspberry Pi e BeagleBone. A utilização de sistemas operacionais baseados em Linux nestes sistemas provê uma série de benefícios, como listados em (KRUGER; HANCKE, 2014):

- Reuso de drivers de rede, sistemas de arquivos e periféricos existentes
- Extensa disponibilidade de pacotes de software prontos, testados e ativamente mantidos
- Sistemas de rede podem ser gerenciados utilizando infraestrutura já familiares do ambiente Linux
- Abstração de software e hardware provida pelo SO permite atualização de pacotes de software independentemente do hardware

Devido à dominância de smartphones utilizando o sistema operacional Android, que também é um sistema baseado no kernel Linux, o Linux possui a maior base instalada de SOs de propósito geral (ASPENCORE., 2017).

Outros exemplos de sistemas embarcados que largamente são baseados em Linux são: switches e roteadores de rede, controladores de automação, televisores, *smartwatches*, consoles de videogame.

Uma discussão mais aprofundada sobre as vantagens e desafios do uso de Linux em sistemas embarcados é apresentada em (SYSTEMS, 2002).

2.4.4 Ferramentas para desenvolvimento de sistemas com Linux Embarcado

Assim como as distribuições Linux existentes para computadores pessoais, existe hoje a disponibilidade de distribuições Linux para plataformas embarcadas. Da mesma forma que as distribuições convencionais, as distribuições embarcadas permitem ao usuário a customização do sistema através da instalação de pacotes de software prontamente disponíveis, como pacotes multimídia, protocolos e serviços de rede, bibliotecas de criptografia, entre outros. Exemplos populares de distribuições Linux embarcadas são o Rasp-

bian (RASPIAN, 2018), uma versão do SO Debian GNU/Linux focado para a SBC Raspberry Pi. Outras distribuições Linux populares de Desktop como o Fedora ou o Arch Linux também provêm variantes voltadas para plataformas embarcadas.

Este tipo de distribuição atende uma parte dos casos de uso de Linux embarcado. Para casos onde é necessário uma maior customização, ou para casos onde não existe uma distribuição disponível para a plataforma ou arquitetura alvo, é necessário se realizar a customização e compilação do kernel Linux e dos pacotes desejados a partir de código fonte. Além das distribuições prontas, existem outros projetos auxiliam essa criação customizada de sistemas Linux embarcado a partir de compilação, conhecidos como *build systems*. Exemplos de *build systems* populares são o Buildroot (BUILDROOT, 2018) e o projeto OpenEmbedded (OPEN EMBEDDED, 2018).

Por fim, existem empresas dedicadas para o desenvolvimento e suporte de Linux para sistemas embarcados. Exemplos de empresas são a MontaVista (MONTAVISTA, 2018), Wind River (RIVER, 2018) e a Enea Software (ENEA, 2018).

3 ESTADO DA ARTE EM SISTEMAS MULTI-AGENTE NA AUTOMAÇÃO

Neste capítulo são apresentados alguns dos trabalhos de pesquisa publicados nos últimos anos na área de Sistemas Multi-Agente. Na Seção 3.1 são apresentados outros trabalhos que fazem revisões do estado da arte do cenário de pesquisa atual. Em seguida são revisados alguns trabalhos em subáreas mais específicas de SMAs com relação a este trabalho. Na Seção 3.2 é apresentada uma tabela comparativa dos trabalhos revisados e são discutidas algumas breves conclusões.

3.1 Cenário atual da pesquisa em Sistemas Multi-Agente

Em (JENNINGS; SYCARA; WOOLDRIDGE, 1998) é feita uma revisão das atividades de pesquisa e desenvolvimento na área de Agentes autônomos e SMAs na época. O autor afirma que essas tecnologia estava começando a encontrar caminhos em produtos comerciais e soluções para o mundo real. No entanto, afirma que ainda restavam muitos desafios de pesquisa e desenvolvimento em aberto, e que soluções robustas e escaláveis ainda deveriam ser encontradas para que sistemas baseados em Agentes pudessem ser uma realidade. É apontado que, no tempo da escrita do trabalho, existiam dois impedimentos técnicos principais para a adoção da tecnologia de Agentes: a falta de metodologias para possibilitar que projetistas especifiquem e estruturem as aplicações como SMAs; e a falta da disponibilidade de kits de ferramentas de desenvolvimento com potencial para a indústria.

(RUDOWSKY, 2004) afirma que "muito trabalho ainda deve ser feito". O trabalho apresenta conceitos e técnicas existentes na época para desenvolvimento de SMAs. São levantados alguns desafios existentes do ponto de vista técnico e social para implementação de SMAs, como: custo, segurança, questões legais/éticas, e aceitação pela sociedade.

Em (PEREIRA; CARRO, 2007), a tecnologia de Agentes é citada como uma das metodologias a ser explorada para o desenvolvimento de sistemas embarcados distribuídos de tempo-real. O principal motivo para considerar a aplicação desses sistemas é o fato de que outras tecnologias não foram capazes de atender todos os requisitos demandados pelos sistemas de automação industriais modernos: integração com sistemas existentes, manipulação cooperativa, processos ágeis, escalabilidade, distribuição, tolerância a falhas, entre outros.

Em (LEITAO; MARIK; VRBA, 2013) é apresentada uma revisão do passado, pre-

sente e previsões para o futuro em SMAs. O autor faz a divisão entre o passado e o presente no ano de 2005, que considera como o fim das aplicações industriais pioneiras com tecnologia de Agentes. O autor considera que apesar das vantagens competitivas com o uso de Agentes, existem ainda problemas que impedem a utilização em larga escala de conceitos de Agentes na indústria, como questionável retorno do investimento, falta de ferramentas de desenvolvimento e padrões, falta de projetos e pessoas especializadas, entre outros. O autor cita que atualmente, as aplicações para a tecnologia de Agentes são de logística e transporte, controle de manufatura, encaminhamento de produtos dinâmico, montagem e aplicações militares e de defesa. De acordo com o autor, o controle de tempo-real ainda é o maior desafio, já que a implementação de sistemas baseados em Agente requer que a indústria faça uma mudança radical na forma como vem projetando e implementando sistemas nas últimas décadas. O autor também considera que existe uma evolução nas plataformas onde os Agentes são executados.

(PEREIRA; RODRIGUES; LEITAO, 2012) aponta que SMAs já são uma tecnologia adequada para desenvolver sistemas que exigem flexibilidade, robustez e reconfigurabilidade. O autor concorda que apesar da adequação dos princípios de SMAs para resolver as necessidades da indústria, a verdadeira implantação de SMAs para aplicações industriais ainda não pode ser ser considerada resolvida. Um dos motivos apontados é a necessidade de se integrar equipamentos de manufatura existentes no sistema. Algumas soluções apontadas são a execução de Agentes em computadores pessoais tradicionais com interface para os equipamentos, ou tratar os Agentes nas próprias máquinas através de, por exemplo, PLCs (programmable logic controllers). O autor sugere que ainda precisam ser ser estabelecidos mecanismos de integração padrões para que seja possível integrar as plataformas industriais no contexto de Agentes, como forma de evitar a prática "caso a caso" que existe atualmente.

(KIRAN; CHANDRAKALA; NAMBIAR, 2017) apresenta uma avaliação da aplicação de SMAs em redes de distribuição de energia elétrica. O modelo tradicional de sistema de gerenciamento de energia não é mais suficiente para o controle de operações do futuro. É apontado que SMA tem as seguintes aplicações neste contexto: controlar a rede levando em consideração fontes renováveis de energia, restrições de voltagem e de temperatura, interação entre entidades para operações de mercado, tomada de decisão de acordo com dados obtidos da rede, extensão de sistemas e funcionalidades existentes, modelagem e simulação. Por fim, os desafios levantados são a aeleção de uma plataforma que permite manutenção a longo prazo, o projeto de cada Agente inteligente, os padrões

de dados e linguagem de comunicação, o nível de segurança na mobilidade de Agentes e a falta de experiência no uso de SMAs.

Outros trabalhos também apresentam revisões do estado da arte em SMA: (MARIK; LAZANSKY, 2007) e (VINYALS; RODRIGUEZ-AGUILAR; CERQUIDES, 2011).

3.1.1 Aplicações de Sistemas Multi-Agente na automação

É de principal interesse deste trabalho artigos a respeito da implementação de SMAs em aplicações industriais e de automação. Esta seção apresenta e discute alguns trabalhos pesquisados nesta área.

Além dos trabalhos apresentados na Seção 3.1, que apontam a automação industrial como uma das principais aplicações de SMAs, o interesse da área também pode ser observado nos seguintes a seguir.

(VRBA et al., 2014) fornece uma visão geral da aplicação atual em sistemas inteligentes de distribuição de energia. Os conceitos considerados por sistemas inteligentes são de SMAs e sistemas orientados a serviço. Entre os principais desafios discutidos que podem ser atendidos pela tecnologia de Agentes estão o equilíbrio de oferta e demanda, mercados de eletricidade, e a manipulação de falhas e diagnósticos. O trabalho apresenta uma lista de diversas aplicações reais de sistemas inteligentes de energia, indicando que os conceitos já estão se tornando realidade nesta área. O trabalho cita a implantação de sistemas embarcados distribuídos utilizando o framework JADE nesta área. O trabalho cita que o uso de sistemas embarcados de dimensões e recursos reduzidos, com baixo custo e baixo consumo de energia são uma tecnologia chave para a criação de um sistema ubíquo de monitoramento e controle de distribuição de energia. Alguns desafios em aberto nesta área são citados: interoperabilidade e padronização entre os projetos, especificação formal e projeto orientado a modelos, e métodos de validação automatizados dos modelos.

A seguir serão apresentados exemplos de aplicações de SMAs na automação.

Em (URE et al., 2014) é apresentado o desenvolvimento e implementação de um sistema autônomo de manutenção de bateria para VANTs. É utilizado um sistema de troca e carga simultânea para compensar o baixo tempo de autonomia dos VANTs. Este sistema é proposto para possibilitar que SMAs compostos de VANTs atendam missões que requerem presença persistente. Experimentos mostram a possibilidade de atender a missões de 3 horas com três VANTs com autonomia de 8-10 minutos por carga.

Existem diversos trabalhos que utilizam o framework JADE para implementação de Agentes.

Em (MASSAWE; KINYUA; AGHDASI, 2010) é apresentada uma implementação de um SMA para gerenciamento de produtos com RFID utilizando JADE. O sistema é composto por Agentes como *leitores*, *geradores de eventos* e *clientes*, utilizando a plataforma de containers do JADE. O trabalho faz uso de Agentes e do JADE devido às características de sistemas distribuídos e comunicação dos Agentes.

A área de redes de distribuição elétrica é uma aplicação bastante explorada para Agentes, isso pode ser mostrado pela variedade de trabalhos na área.

O trabalho em (GHOSN et al., 2010) aponta que redes de distribuição elétrica são complexas pois são sistemas dinâmicos que podem ser inseguros, não confiáveis e ineficientes para servir os clientes. A promessa das redes de distribuição inteligentes é desenvolver redes distribuídas capazes de monitorar e controlar automaticamente o desempenho da rede.

(KULASEKERA et al., 2011) reforça que a aplicação de SMAs em sistemas elétricos está se tornando incrementalmente popular. O autor sugere que os frameworks atuais possuem limitações no balanceamento de carga e negociação de múltiplos objetivos. O autor sugere que o foco do desenvolvimento deve ser no desenvolvimento de frameworks e arquiteturas com capacidades melhoradas para suprir essas limitações.

As características de sistema distribuído, flexibilidade e robustez dos SMAs e do JADE também são explorados em (ESTRADA; LEE, 2013). Neste trabalho, o JADE é utilizado para implementar um sistema de controle de usinas elétricas. Neste trabalho também são utilizadas outras tecnologias em conjunto com o JADE, como o MATLAB®.

Alguns artigos abordam a implementação de Agentes em "prédios inteligentes".

Em (DAVIDSSON; BOMAN, 2000), os Agentes são componentes que fazem parte do prédio, como aquecedores ou lâmpadas existentes nas salas do prédio, ou outros dispositivos como computadores e cafeteiras. O sistema utiliza a própria rede de alimentação elétrica do prédio para comunicação, evitando a necessidade de criar uma nova rede. O objetivo do uso do SMA é controlar esses dispositivos de forma inteligente, levando em conta as necessidades dos seus usuários, de forma a aumentar o conforto e também gerenciar o consumo de energia. Através de simulações, é estimado que se possa ter uma economia de 40% no consumo do sistema de controle de temperatura do prédio através do uso do SMA, quando comparado com o controle manual pelo termostato.

(ROSCIA; LONGO; LAZAROIU, 2013) discute o uso de SMAs no contexto de

uma “cidade inteligente” e também no contexto da IOT. É proposta uma arquitetura com os Agentes para diversas atividades da cidade inteligente, e é proposta a implementação utilizando o framework ZEUS (NWANA et al., 1999). Além disso, é descrito em detalhe como cada elemento do sistema é especificado e a comunicação entre eles.

(MUSTAPHA; MCHEICK; MELLOULI, 2013) propõe um framework metodológico para desenvolvimento de sistemas complexos utilizando SMAs. A principal contribuição do framework é refletir a estrutura organizacional e a organização dos Agentes na simulação. Exemplos de sistemas complexos considerados pelo trabalho são redes de distribuição e controle de desastres naturais. A metodologia propõe a descrição do sistema através de modelos que são transformados sucessivamente, e em seguida é possível fazer simulações.

(KAINDL; VALLEE; ARNAUTOVIC, 2013) descreve uma aplicação de SMA para um sistema de transporte de carga através de *pallets*. O autor propõe que cada Agente nesse sistema possua uma identificação própria e possa se auto reconfigurar a partir dessa identificação. Cada Agente possui também um modelo de si mesmo para que seja possível monitorar e detectar falhas em si mesmo. O sistema proposto é voltado para detecção de falhas, e o trabalho descreve comportamentos esperados quando alguns tipos de falha são detectados.

Existem também trabalhos com exemplos de aplicações com Agentes em hardware, na área de automação.

(NAJI, 2005) apresenta um exemplo de implementação de *fusão de sensores* utilizando Agentes em hardware. Fusão de sensores consiste de fazer medidas com mais de um sensor, de forma redundante, de forma a obter um resultado final. No caso mais simples isso consiste da média entre os sensores, porém podem haver casos onde existe divergência na confiabilidade de cada sensor. O trabalho propõe que um sistema composto de Agentes para fazer a fusão de sensores pode ter melhor performance do que implementações tradicionais. Os resultados dessa implementação de Agentes em hardware apresentam ganhos nos resultados com essa abordagem.

Em (MENG, 2006) é proposto um modelo que utiliza Agentes para simplificar o projeto do sistema e integrar de forma eficiente Agentes em hardware e Agentes em software. Esta metodologia é aplicada para projetar um robô móvel, sendo utilizada para projetar aspectos de alto nível do sistema. O sistema é primeiro decomposto em vários Agentes com base nas especificações, considerando que estes Agentes podem realizar tarefas específicas de forma independente e podem se comunicar uns com os outros. Estes

Agentes então são particionados em Agentes de software e Agentes de hardware. Considerando uma aplicação onde o robô deve se movimentar numa sala com obstáculos em direção a um alvo, são observados ganhos consideráveis com essa divisão com relação a uma plataforma composta somente de software.

(SCHUTZ et al., 2011) apresenta um sistema composto de Agentes para controle de sistemas de produção. Este trabalho apresenta uma solução capaz de atender requisitos de tempo-real através da implementação de Agentes em um PLC. O autor cita que poucos trabalhos utilizam a abordagem de implementação de Agentes em PLC.

(BELKACEMI et al., 2011) descreve o desenvolvimento e implementação de um sistema de controle de redes de distribuição elétrica baseado em Agentes em hardware. A tecnologia de SMAs é utilizada para auxiliar na distribuição do processamento e do controle, além de que a comunicação entre os Agentes possibilita a coordenação e troca de dados para alcançar uma melhor solução global.

(PEIXOTO; PEREIRA, 2016) afirma que a indústria carece de sistemas produtivos que atendam a diversidade de demanda do mercado, com eficiência e capacidade de adaptação rápida. As novas soluções para estas necessidades remetem ao uso de novas tecnologias de automação. Porém, a maioria dos sistemas integrados de manufatura implantados em indústria possui seu gerenciamento por PLCs, assim como suas interligações elétricas e lógicas de controle. O uso de SMA em PLCs propicia os requisitos de diversidade, agilidade e auto-organização no meio produtivo. O método proposto analisa o PLC e descreve uma rotina de ações a serem seguidas para que ele se insira em SMA, valendo-se das funcionalidades que um agente pode oferecer. Sua análise de aplicação ocorreu em três cenários distintos, onde a abordagem por sistemas auto-organizáveis apresentou melhores resultados para atingir os requisitos de diversidade, agilidade e auto-organização, a partir de sistemas multiagentes interagindo com os PLCs e suas lógicas locais.

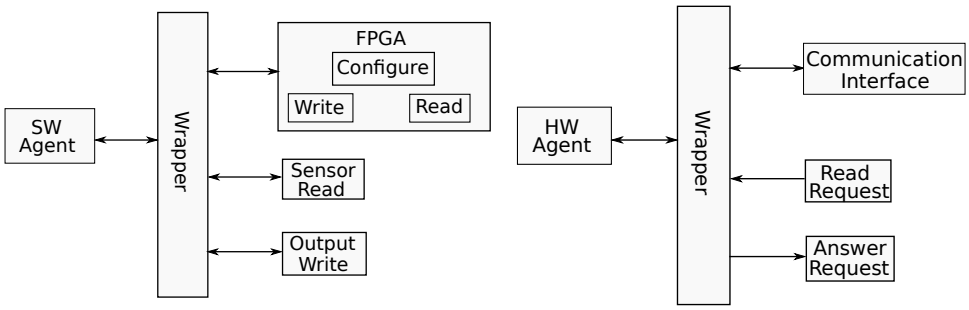
3.1.2 Agentes em Hardware e Agentes Reconfiguráveis

É do interesse especial deste trabalho artigos que mencionem implementação de Agentes em hardware ou em arquiteturas reconfiguráveis. Esta seção apresenta e discute alguns trabalhos pesquisados nesta área.

Em (CEMIN; PEREIRA, 2012)(CEMIN; GOTZ; PEREIRA, 2012) é apresentado uma arquitetura de Agentes para RSSF (Redes de Sensores Sem Fio) onde Agentes podem estar implementados em software ou em hardware. É utilizado o framework JADE e

a execução em hardware ou em software não é percebida pelo mundo externo, sendo possível a comunicação dos Agentes em hardware com os Agentes em software. Os Agentes em hardware são implementados em FPGAs. A contribuição deste trabalho é que com esta arquitetura, é possível se fazer a migração de Agentes implementados em hardware para software ou vice-versa, em tempo de execução. Para isto, o trabalho desenvolve um serviço denominado *wrapper*, que realiza a abstração entre os Agentes em software e em hardware. A Figura 3.1 mostra a implementação do *wrapper*.

Figura 3.1 – Serviços oferecidos pelo Wrapper aos Agentes em hardware e software em (CEMIN; PEREIRA, 2012)



Fonte: (CEMIN; PEREIRA, 2012)

Os resultados discutem o tempo de migração de Agentes entre software e hardware e alguns resultados de comparação entre tempo de execução entre software e hardware. Pelos resultados apresentados e pelos trabalhos futuros sugeridos, ainda pode ser melhor explorada e otimizada a possibilidade de migração de Agentes entre software e hardware. Uma das formas é a utilização de reconfiguração parcial de FPGA, que um tema citado no trabalho porém a implementação é deixada como proposta de extensão e trabalhos futuros.

O trabalho (NAJI, 2003) discute sobre a implementação de operações *data flow* utilizando Agentes em hardware. Em uma operação *data flow*, a execução de cada operação é dirigida pela disponibilidade de dados na entrada da operação. A saída de cada operação pode ser conectada à entrada de uma operação seguinte, formando um grafo *data flow*. O trabalho mostra formas de organizar a implementação no hardware utilizando Agentes para cada operação do grafo.

(SCHNEIDER; NAGGATZ; SPALLEK, 2007) propõe uma implementação de arquitetura com Agentes em hardware. Nesta arquitetura, existe um serviço gerenciador chamado *amsys*, que gerencia os Agentes disponíveis na plataforma. Cada Agente em hardware possui disponível lógica programável (FPGA) para a implementação da parte

operativa do Agente e um microcontrolador associado que se comunica com o *amsys* e gerencia a parte operativa. A arquitetura considera também problemas como autodiagnóstico e a possibilidade de usar múltiplos controladores por Agente em hardware para melhorar a resiliência do sistema.

O trabalho em (MOIS et al., 2010) endereça mais especificamente o problema de auto-teste e reparo de Agentes em hardware em sistemas distribuídos. O objetivo é obter um sistema reconfigurável que não necessite de intervenção humana para teste e manutenção. O trabalho utiliza Agentes que se comunicam através de redes sem-fio. A arquitetura proposta faz a implementação de um Agente monitor em hardware que fica constantemente monitorando e executando testes nos Agentes do sistema. Caso o teste em um Agente do sistema não retorne o valor esperado, o Agente monitor pode reconfigurar o Agente falho para que volte a operar corretamente.

(VRBA; MARIK, 2010) também discute o assunto de reconfiguração de Agentes na automação. Dois pontos principais são discutidos, considerando SMAs com aplicação em sistemas de transporte: A reconfiguração automática de caminhos para transporte, e a possibilidade de adição e remoção de elementos de forma dinâmica. São levados em conta limitações de sistemas atuais, onde para se fazer as alterações citadas, é necessário realizar manutenções manuais no sistema. São propostas soluções para os problemas apresentados e uma implementação para validar as soluções.

Em (LIU, 2013) é apresentada uma implementação de SMA com número de Agentes configuráveis implementados em um mesmo FPGA. A aplicação do sistema é um sistema com Agentes para encontrar soluções no jogo de estratégia *Blokus Duo*. O autor não apresenta resultados de comparação do tempo de execução com relação à quantidade de Agentes configurados, apenas mostra a prova de conceito e mostra que a implementação em FPGA possui desempenho amplamente superior a uma implementação em software.

Outras propostas de arquiteturas reconfiguráveis, com comparações de desempenho, são revisadas em (SURESH; HENRY; RANGARAJAN, 2009).

3.1.3 Sistemas Multi-Agente com requisitos de tempo-real

Em (KROL; NOWAKOWSKI, 2013) e (FILGUEIRAS; LUNG; RECH, 2012) foram realizados estudos a respeito da utilização do JADE em aplicações com requisitos de tempo-real. É apontado que ao contrário do que se possa esperar, existe pouco esforço de pesquisa dedicado para o suporte a tempo-real em SMAs, mesmo que essa integração

seja evidente.

(KROL; NOWAKOWSKI, 2013) explora o uso do JADE em conjunto com o RTSJ (Real-Time Specification for Java) numa aplicação de exemplo considerando um Agente capaz de dirigir um automóvel. O trabalho realiza a comparação de implementações do Agente utilizando a API de threads e a API utilizando threads de tempo-real. Um dos principais obstáculos apontados é o fato de que o escalonamento de comportamentos do JADE é não-preemptivo, tornando difícil incluir deadlines ou prioridades. O trabalho conclui que JADE não escala em conjunto com o RTSJ. Pelos resultados apresentados, o tempo de execução aumenta significativamente se o número de Agentes for aumentado, e a integração entre as duas APIs apresenta alta complexidade.

Em (FILGUEIRAS; LUNG; RECH, 2012) é proposta uma camada de software para adicionar suporte a escalonamento de tempo-real no JADE, denominado RT-JADE. O trabalho consiste de uma arquitetura de Agentes onde existe um Agente servidor que aloca e escalona tarefas para Agentes móveis. O requisito de tempo-real é atingido através da política de escalonamento da execução dessas tarefas, que leva em conta deadlines e níveis de prioridades. Foram experimentados seis algoritmos de escalonamentos diferentes e diferentes tempos de deadline para as tarefas. Os resultados mostram que com o uso do RT-JADE o desempenho de tempo-real do framework é melhor do que o JADE com a política de escalonamento padrão. Em casos específicos, foi observado que é possível atingir 100% dos deadlines cumpridos.

Em (PERRAJU, 1999) discute-se o uso de SMAs em sistemas que possuem alta garantia de funcionamento como requisito. Uma das vantagens do SMA neste ambiente é que os múltiplos Agentes podem colaborar e tomar papéis para assumir um dado objetivo durante a missão. Através da comunicação, eles podem negociar qual o melhor papel que cada um deve assumir para facilitar o objetivo. Modelando cada Agente através de um modelo composto por duas camadas – a camada deliberativa e a camada reativa – e modelando o sistema através de um autômato, o autor mostra que o sistema modelado satisfaz os requisitos para completar a missão.

3.1.4 Outros trabalhos relacionados

Em (VYROUBAL; KUSEK, 2013) são propostas soluções para se fazer migração de Agentes JADE entre plataformas com máquinas virtuais Java tradicionais e a máquina Dalvik do sistema operacional Android. Por possuírem bytecodes incompatíveis, é ne-

cessário se fazer a conversão de bytecodes para a migração do código dos Agentes entre as duas plataformas. O artigo considera a implementação JADE-LEAP (Java Agent Development Framework-Lightweight Extensible Agent Platform) do JADE, que é uma implementação para sistemas embarcados que não implementa a migração de Agentes. São propostas várias técnicas para contornar este problema, e são também descritas várias alternativas para o problema da tradução dos bytecodes entre as duas plataformas. O trabalho não apresenta resultados ou experimentos comparando as técnicas propostas. Este trabalho mostra que existe envolvimento na área de Agentes também com as plataformas móveis atuais.

(GODDEMEIER; BEHNKE; WIETFELD, 2013) analisa a influência das condições de comunicação e da topologia de rede na chegada a um consenso por diversos Agentes em um SMA. É proposto um modelo para determinar o número ótimo de links de comunicação entre Agentes a fim de encontrar o melhor equilíbrio entre a cobertura e velocidade de convergência. São mostradas simulações levando em conta a quantidade de erros de comunicação e esses resultados são utilizados para auxiliar a decisão sobre a topologia de um sistema composto por VANTs. De acordo com os resultados experimentais, uma pequena quantidade de erros já pode impossibilitar a chegada a um consenso utilizando algoritmos conhecidos.

(BOSSE, 2014) propõe a criação de SMAs onde cada Agente é implementado sobre o hardware dedicado, utilizando uma arquitetura *agent-on-chip* (AoC). A proposta consiste de mapear o comportamento, interação e conceitos de migração entre Agentes em microchips. Cada AoC possui disponíveis lógica reconfigurável e máquinas de estado programáveis para que seja feita a implementação do Agente. O trabalho propõe a arquitetura e trata questões como a síntese de Agentes para essa arquitetura. É apresentado um estudo de caso aplicando a teoria em sistemas de monitoramento de saúde.

3.2 Considerações finais

Na Tabela 3.1 é apresentado um resumo comparativo dos trabalhos pesquisados.

É observado em concordância com as observações dos trabalhos estudados, que poucos trabalhos tratam o problema de controle de tempo-real. A resiliência a falhas é um requisito naturalmente explorado pela aplicação de SMAs, no entanto boa parte dos trabalhos estudados não atacam esse requisito diretamente, sendo o foco em outros requisitos como o de sistemas distribuídos ou desempenho.

Por fim, é observado que poucos trabalhos realizam integração de Agentes em hardware com Agentes em software, sendo que essa integração ainda pode ser melhor explorada. Em particular, o tema de reconfiguração parcial é pouco abordado. Em (CEMIN; PEREIRA, 2012)(CEMIN; GOTZ; PEREIRA, 2012) é abordado o tema de reconfiguração parcial, porém não é realizada uma implementação para medir os resultados. Sendo assim, o uso de reconfiguração parcial em conjunto com a implementação de Agentes em hardware pode ser melhor explorada.

O *wrapper* apresentado no trabalho de referência possui limitações com relação à forma de acesso ao FPGA. Esta limitação existe pois é utilizado um processador externo ao FPGA e são descritos somente acessos no formato MMIO. Além disso, sua implementação não é está disponível abertamente. Por isso, este trabalho apresenta uma nova abordagem, descrita no Capítulo 4.

Tabela 3.1 – Comparação de trabalhos relacionados

Trabalho	Agentes em Software	Agentes em Hardware	Migração de Agentes	Tempo-real
Vrba et al. (2014)	•			
Ure et al. (2014)	•			
Massawe, Kinyua e Aghdasi (2010)	•			
Ghosn et al. (2010)	•		•	
Kulasekera et al. (2011)	•		•	
Estrada e Lee (2013)	•			
Davidsson e Boman (2000)	•			
Roscia, Longo e Lazaroiu (2013)	•			
Mustapha, Mcheick e Mellouli (2013)	•			
Kaindl, Vallee e Arnautovic (2013)	•			
Naji (2005)	•	•		
Meng (2006)	•	•		•
Schutz et al. (2011)		•		•
Belkacemi et al. (2011)		•	•	
Peixoto e Pereira (2016)	•	•		
Cemin e Pereira (2012)	•	•		
Naji (2003)		•		•
Schneider, Naggatz e Spallek (2007)		•		•
Mois et al. (2010)		•		•
Vrba e Marik (2010)		•		•
Liu (2013)	•			
Krol e Nowakowski (2013)	•		•	
Filgueiras, Lung e Rech (2012)	•		•	
Perraju (1999)	•			
Vyroubal e Kusek (2013)	•		•	
Goddemeier, Behnke e Wietfeld (2013)	•			
Bosse (2014)		•	•	
Este Trabalho	•	•	•	

Fonte: Autor

4 PLATAFORMA PARA AGENTES EM HARDWARE UTILIZANDO RECONFIGURAÇÃO PARCIAL

Neste capítulo estão descritos de forma detalhada os componentes de hardware e software utilizados para a criação da plataforma proposta e sua interação para compor uma plataforma de Agentes. Na Seção 4.1 é apresentada uma visão geral da arquitetura da plataforma. Na Seção 4.2 e na Seção 4.3 são apresentados detalhamentos dos componentes de hardware e software da plataforma, respectivamente. Na Seção 4.4 é apresentada uma proposta de Agentes auxiliares para a implementação de reconfiguração parcial na plataforma. Por fim, na Seção 4.5 são brevemente discutidas as contribuições observadas na arquitetura proposta e na plataforma.

4.1 Arquitetura

Este trabalho propõe uma plataforma capaz de permitir a implementação de Agentes com capacidades de processamento de alto desempenho em hardware através do uso de lógica programável e reconfiguração parcial de hardware.

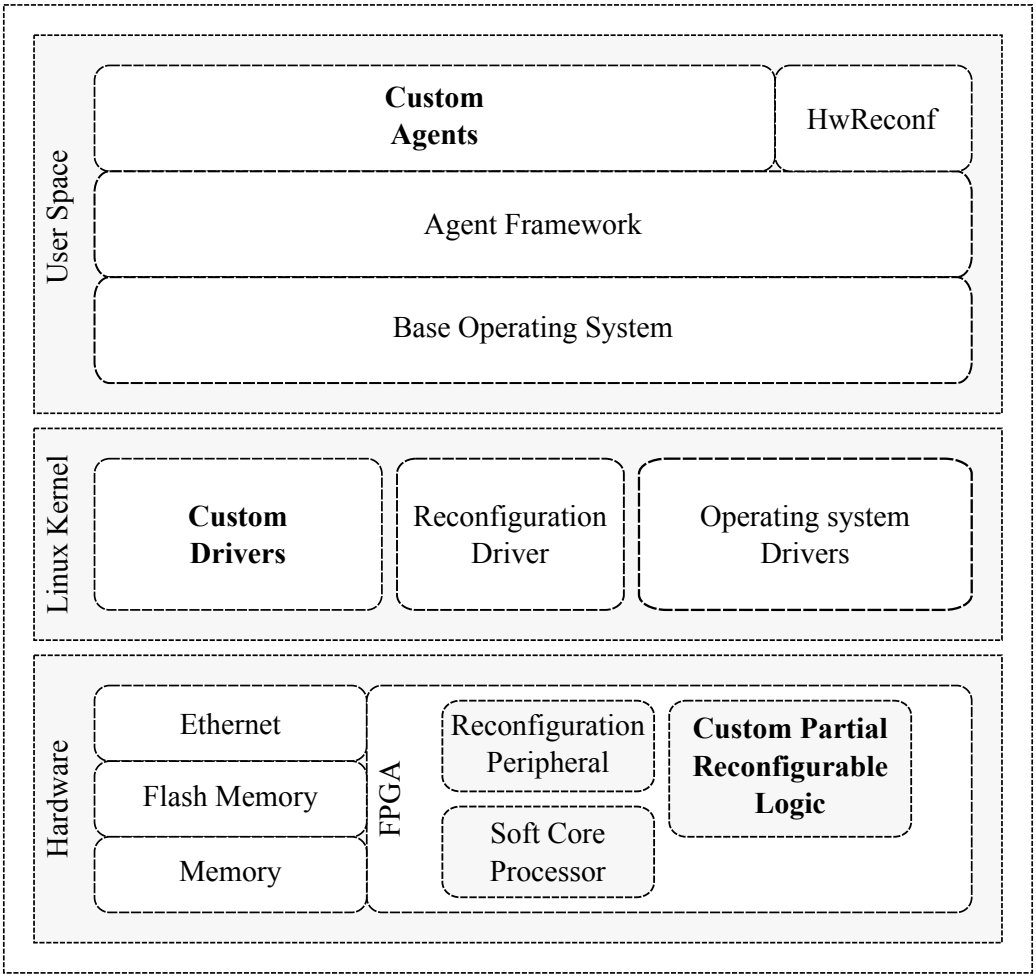
A arquitetura de software da plataforma é composta de um framework de Agentes, um sistema operacional e drivers associados para as implementações em hardware. A arquitetura de hardware é composta de um processador principal capaz de executar o framework de Agentes, e de recursos de lógica reconfigurável.

A Figura 4.1 mostra uma visão geral desta arquitetura, mostrando os componentes mais importantes. Na Figura 4.1, os componentes que podem ser customizados pelo usuário são marcados em negrito.

No projeto da plataforma, são desenvolvidos componentes em níveis de lógica programável, nível de kernel do SO (kernel) e no nível de espaço de usuário. Por necessitar de projeto em várias camadas, são utilizadas diversas ferramentas para automatizar e facilitar o projeto, tanto em nível de software quanto em nível de hardware.

Na camada de hardware, os principais requisitos são de reconfigurabilidade do hardware, ao mesmo tempo em que oferecer um ambiente para execução dos Agentes, e bom desempenho de interação entre o framework de Agentes e as funções implementadas em hardware. Os detalhes da implementação de referência em hardware são apresentados na Seção 4.2.

Figura 4.1 – Visão geral das camadas da arquitetura da plataforma, mostrando os componentes mais importantes – áreas marcadas em negrito podem ser customizadas pelo usuário



Fonte: Autor

Nas camada de software (SO e aplicação), os principais objetivos são de oferecer uma plataforma de Agentes, com uma interface de Agentes padronizada e suportada.

Os framework de Agentes é a interface da plataforma com o mundo externo. O sistema embarcado necessita de um sistema Linux otimizado para dar suporte à execução dos Agentes e comunicação com o hardware. Para geração de um sistema Linux final otimizado, a plataforma necessita de ferramentas para geração de sistema Linux embarcado, conforme introduzido na Seção 2.4.4. Os detalhes da implementação de referência em hardware são apresentados na Seção 4.3.

A arquitetura apresentada na Figura 4.1 pode ser implementada utilizando diversas formas e tecnologias. Neste trabalho, foi realizada uma implementação desta arquitetura, resultando na plataforma para agentes em hardware utilizando reconfiguração parcial.

A Tabela 4.1 apresenta a correspondência entre as camadas da arquitetura, a tecnologia utilizada na implementação de referência da plataforma.

Tabela 4.1 – Correspondência entre arquitetura e implementação de referência da plataforma.

Arquitetura	Plataforma
Agentes customizados	Definido pelo usuário
HwReconf	HwReconfAgent
Framework de Agentes	JADE
Sistema operacional Base	Buildroot, JamVM
Drivers customizados	Definido pelo usuário
Driver de reconfiguração parcial	Driver HWICAP
Periférico de reconfiguração parcial	HWICAP
Processador soft core	MicroBlaze
Sistema operacional	Linux
Hardware reconfiurável	Definido pelo usuário

Fonte: Autor

O restante deste capítulo descreve a plataforma considerando a implementação de referência. Ou seja, mesmo que sejam possíveis diversas implementações da arquitetura da Figura 4.1, algumas características são discutidas considerando a tecnologia escolhida para implementação.

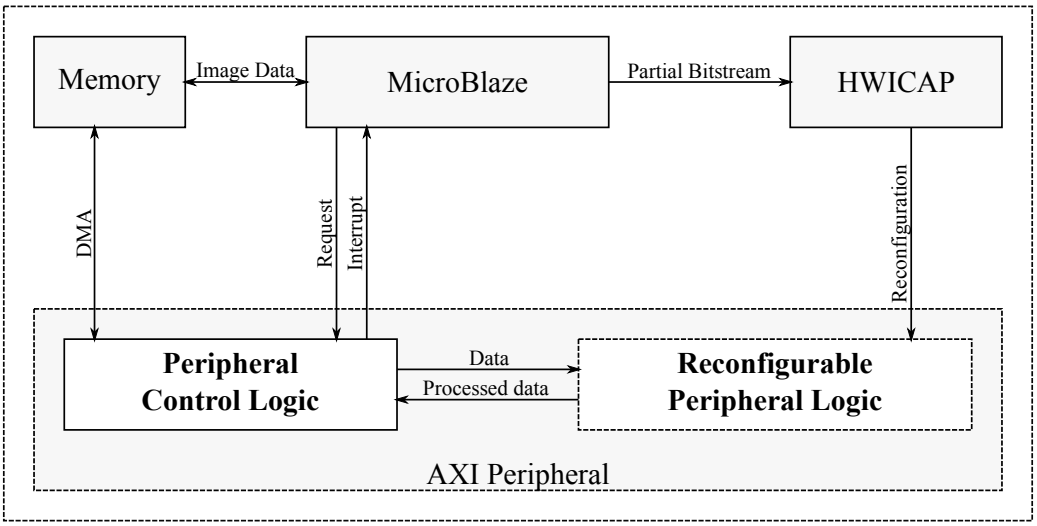
4.2 Componentes de hardware

A Figura 4.2 mostra uma visão detalhada da plataforma dentro da lógica programável, onde um Agente em hardware pode estar implementado.

A plataforma faz uso de um processador soft core devido à possibilidade de maior integração com a lógica programável. Na figura, é representado pelo bloco MicroBlaze. O MicroBlaze faz uso da memória externa disponível no hardware para execução do sistema operacional, e armazenamento dos dados a serem processados nos blocos de hardware. O processador soft core MicroBlaze é apresentado em detalhes e discutido na Seção 4.2.1.

O barramento de comunicação entre o processador e os periféricos customizados é apresentado na Seção 4.2.2. Este barramento é utilizado devido ao alto desempenho que oferece e para tomar vantagem da infraestrutura de ferramentas que existe para esta solução. A arquitetura proposta para os periféricos customizados é apresentada na Seção 4.2.3.

Figura 4.2 – Diagrama de blocos dos componentes de hardware – com exceção da memória, todos os componentes são instanciados dentro da lógica programável



Fonte: Autor

O bloco HWICAP é um componente importante na plataforma, responsável por possibilitar o uso de reconfiguração parcial em conjunto com o MicroBlaze. Este bloco é discutido em mais detalhes na apresentado na Seção 4.3.3.

4.2.1 Processador soft core

Na arquitetura, se propõe o uso de um processador soft core dentro da lógica programável. Apesar de os processadores soft core em geral não oferecerem desempenho comparável a processadores embarcados presentes em SOCs modernos, o uso de processadores soft core para esta aplicação é interessante pois o requisito mais importante é que ele seja eficiente em prover comunicação entre o framework de Agentes e os recursos de lógica programável. Por serem mais facilmente integrados à lógica programável e consumirem menos recursos, os processadores soft core se apresentam como uma escolha interessante para esta aplicação.

Esta plataforma foi baseada no uso do processador MicroBlaze. MicroBlaze é um núcleo processador soft core disponibilizado pela Xilinx, que possui diversas opções de customização para se adequar a uma determinada aplicação. O MicroBlaze é acompanhado de ferramentas que auxiliam a sua extensão através de periféricos customizados. A customizabilidade e o suporte a ferramentas de customização avançadas como o Xilinx

EDK (Embedded Development Kit) e o Xilinx Vivado tornam o MicroBlaze uma escolha interessante para esta plataforma. Outros atrativos para utilização do MicroBlaze com relação a outros processadores soft core é o suporte a versões atualizadas de Linux e suporte a execução do framework de Agentes JADE, conforme será discutido na Seção 4.3.

Cabe lembrar que existem hoje diversas implementações de processadores que podem ser instanciados em lógica programável. Dentro destes processadores disponíveis, vários também são capazes de executar SOs modernos e frameworks de Agentes, conforme apresentado na Seção 2.3.3. Além disso, existem também hoje dispositivos de lógica programável que contém instâncias hard core de processadores e que oferecem capacidades semelhantes, conforme discutido na Seção 2.3.4. Este tipo de dispositivo também pode ser utilizado para uma implementação da arquitetura.

Dentre as opções de customização do MicroBlaze, existe a possibilidade de ajustar parâmetros que podem aumentar o seu desempenho, como frequência de operação, tamanhos de memória cache e suporte a operações de ponto flutuante. Apesar disso, mesmo com o processador configurado para priorizar o desempenho, o MicroBlaze oferece capacidade limitada para aplicações de processamento intensivas como algoritmos de processamento de sinais e imagem. No entanto, isso é compensado pelas facilidades de interfaceamento com os recursos de lógica programável, permitindo que estas aplicações sejam delegadas para a lógica programável de forma mais eficiente.

A Figura 4.3 provê uma representação da configuração utilizada para o processador MicroBlaze nesta plataforma.

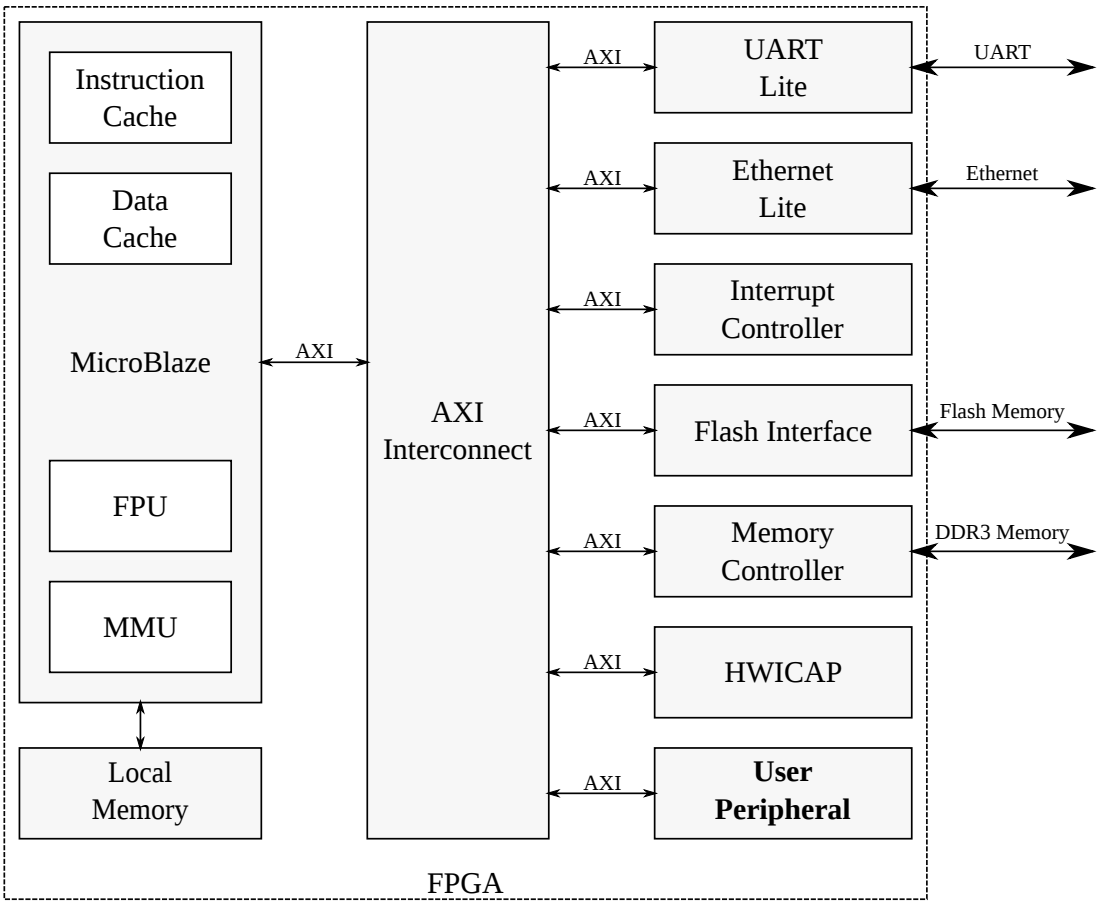
4.2.2 Barramento de comunicação com o hardware

AXI (Advanced eXtensible Interface) é uma parte do ARM AMBA (Advanced Microcontroller Bus Architecture), uma família de barramentos para microcontroladores introduzida inicialmente em 1996 (XILINX, 2011). A primeira versão do AXI foi incluída no AMBA 3.0, lançado em 2003. O AMBA 4.0, lançado em 2010, inclui uma segunda versão do AXI, o AXI4.

Existem três tipos de interfaces AXI4:

- AXI4, para aplicações com requisitos de alto desempenho controlada por comunicação mapeada em memória.
- AXI4-Lite, para aplicações simples, de baixo volume de dados e controlada por

Figura 4.3 – Diagrama de blocos dos componentes do sistema MicroBlaze



Fonte: Autor

comunicação mapeada em memória (por exemplo, leitura e escrita de registradores de controle e status).

- AXI4-Stream, para aplicações de streaming de dados.

As ferramentas Xilinx possuem suporte e ferramentas para auxiliar a criação de periféricos AXI4. Este é o padrão de interface preferencial ao se utilizar soluções da Xilinx para implementação de sistemas embarcados.

Além disso, existem outros benefícios na utilização do AXI4 para implementação de periféricos. Por se tratar de um padrão de interface, também utilizado por processadores ARM, periféricos implementados desta forma podem ser reutilizados em outros projetos e ferramentas.

As interfaces AXI4 e AXI4-Lite também permitem a implementação de periféricos alto desempenho, através do uso de periféricos capazes de se tornar mestres no barramento, realizando operações de DMA. Dentre estes dois tipos de interfaces, o dife-

rencial do AXI4 com relação ao AXI4-Lite é que o AXI4 suporta adicionalmente acessos em rajada à memória.

Da mesma forma que a escolha do uso do processador MicroBlaze, outros tipos de barramentos com capacidades semelhantes podem ser utilizados para a implementação da arquitetura. Um exemplo de outro barramento com funcionalidades semelhantes é o AVALON (Intel Corporation, 2017).

4.2.3 Periféricos customizados

Um dos principais objetivos da plataforma é suportar a comunicação eficiente entre o processador e periféricos customizados implementados na lógica programável, para acelerar as funções necessárias para os Agentes.

Através do uso do Xilinx EDK, periféricos customizados podem ser conectados ao barramento de periféricos do MicroBlaze. As versões modernas do MicroBlaze, suportam interfaceamento nativo com diferentes variações do barramento AXI. AXI é parte da especificação AMBA e é um padrão atual para implementação de periféricos em SOCs.

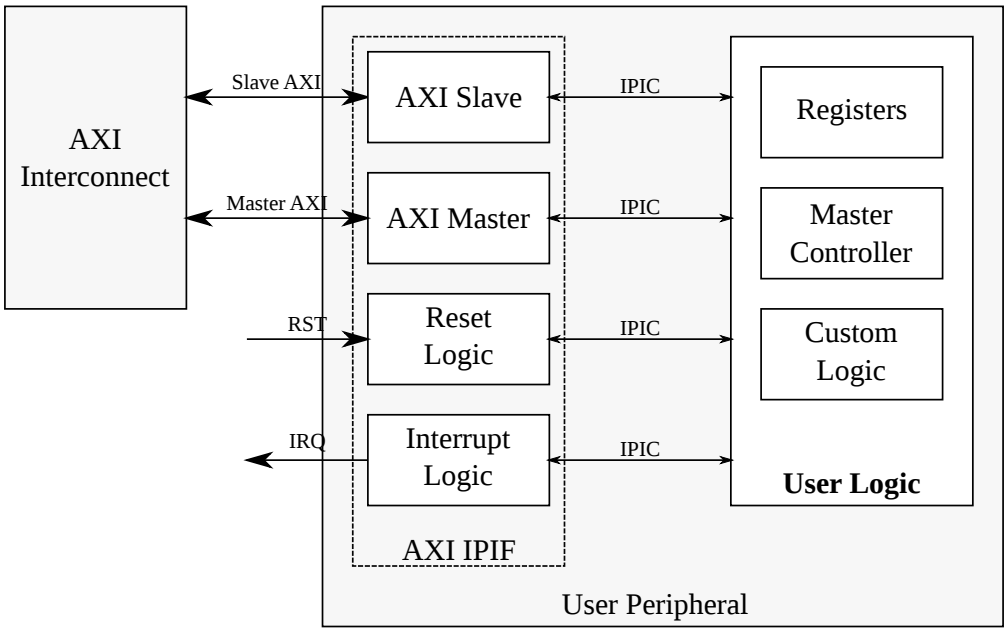
A Figura 4.4 mostra o exemplo de um periférico implementado para esta plataforma. Mais detalhes sobre funcionamento destes periféricos serão discutidos no decorrer desta seção.

Para que seja possível realizar processamento de imagens de alta resolução com alto desempenho, que é um dos objetivos apresentados para a plataforma, é necessário que seja possível transmitir dados do processador principal para a lógica programável de forma eficiente. A alternativa de menor complexidade seria implementar a transferência utilizando MMIO (ou PIO). No entanto, a forma preferida para transmitir dados com este fim é através de transferências DMA em conjunto com uma interrupção do periférico, conforme discutido na Seção 2.3.6, que faz um uso mais eficiente do barramento de transmissão de dados e não requer intervenção do processador após o início da transferência.

A Figura 4.5 mostra em mais detalhes o funcionamento do controlador de DMA instanciado dentro dos periféricos.

A implementação da funcionalidade de DMA pode ter impacto no software, conforme é discutido também na Seção 4.3.2. A forma mais simples de se implementar suporte a DMA no hardware é através de endereçamento contíguo, de forma que o hardware tenha somente que realizar as escritas na memória em sequência. No entanto, se o processador estiver utilizando um SO com acesso à memória como memória virtual atra-

Figura 4.4 – Diagrama de blocos de um periférico customizado, quando gerado através do EDK



Fonte: Autor

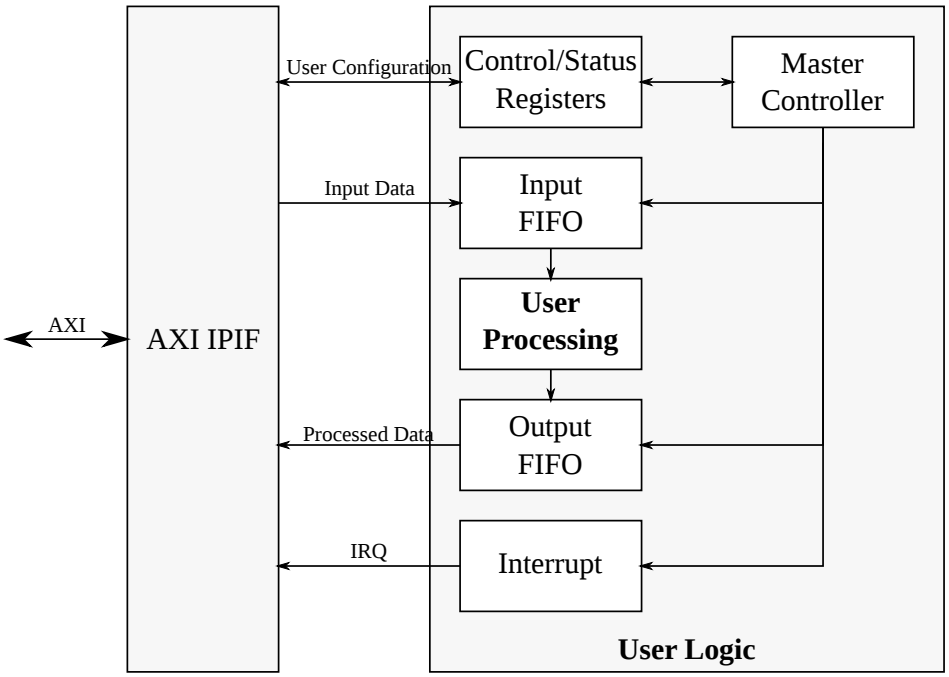
vés de uma MMU, o software terá de prover uma forma de acessar a memória fisicamente contígua após a operação.

Uma alternativa ao uso de grandes buffers de DMA é a implementação de funcionalidade *scatter/gather* no hardware. *scatter/gather* é uma funcionalidade em hardware que utiliza um descritor em memória contendo endereços de um conjunto de buffers menores para onde se pode fazer DMA. A vantagem desta funcionalidade é que neste caso não existem as dificuldades de obtenção de um buffer contíguo fisicamente.

Outro ganho de performance pode ser obtido se o hardware for projetado para que o formato de dados coincida com o formato disponibilizado pelas biblioteca de software. Por exemplo, em uma aplicação de processamento de imagem, o hardware pode ser projetado para utilizar a mesma quantidade de bits por componente de cor, e de forma que os componentes de cor estejam disponibilizados na mesma ordem. Se o periférico de hardware for construído dessa forma, isso elimina trabalho de preprocessamento de dados que teria que ser feito em software para prover os dados de forma que o hardware consiga processar. Este projeto em conjunto torna a implementação em hardware mais simples e eficiente.

Conforme mostrado na Figura 4.1, é necessário que exista também um driver para comunicação com o periférico de hardware do usuário. Dependendo dos tipos de disposi-

Figura 4.5 – Diagrama de blocos da unidade de processamento do periférico customizado, que contém o processamento do usuário



Fonte: Autor

tivos, pode ser utilizado um driver compartilhado entre variações de um tipo de periférico. A implementação deste driver é discutida na Seção 4.3.2.

4.3 Componentes de software

Apesar de a camada de SO não ser de particular interesse para a aplicação final de Agentes, ela é de fundamental importância para que seja possível fazer a comunicação entre os Agentes e a lógica customizada. Como o hardware implementado é customizado e pode oferecer interfaces variadas para o software, é necessária a implementação de drivers no Linux. O modelo de drivers proposto é apresentado na Seção 4.3.2. O objetivo deste modelo é oferecer o melhor desempenho para as aplicações, provendo a infraestrutura para que o software consiga atender o requisito de comunicação eficiente com o hardware programável.

O uso do JADE é discutido na Seção 4.3.5. A funcionalidade de migração de Agentes é de especial interesse deste trabalho, devido à oportunidade de integração com a funcionalidade de reconfiguração parcial do hardware. Por ser uma plataforma com

recursos limitados e baixo poder de processamento em software, é utilizada uma implementação reduzida de JVM para dar suporte à execução do JADE. Nesta plataforma, o ambiente Java é provido pela JVM JamVM.

Conforme discutido na Seção 2.4.4, em alguns casos na criação de sistemas Linux embarcado, é vantajoso (ou necessário) se construir um sistema customizado para o alvo embarcado. Por se utilizar o MicroBlaze na plataforma, devido aos seus recursos limitados e à demanda de software especializado (como a JamVM), é utilizado o *build system* Buildroot. O seu uso e características do sistema base são discutidos na Seção 4.3.4.

4.3.1 Linux Kernel

Para facilitar o desenvolvimento do software em mais alto nível e também devido à dependência do uso do framework JADE, a plataforma utiliza o Linux como SO. O Linux suporta a grande maioria das arquiteturas de processador utilizadas em sistemas embarcados, incluindo processadores soft core como o MicroBlaze ou Nios2.

Com o processador executando um SO como o Linux, é possível acessar estes dispositivos conectados no barramento AXI através de drivers no kernel do Linux. É necessário se utilizar drivers pois aplicações de usuário no Linux não são capazes de acessar diretamente os registradores mapeados em memória, dentre outras limitações como o registro de interrupções ou alocação de buffers DMA.

A versão do kernel Linux utilizada como referência nesta implementação é 3.18, com modificações realizadas pela Xilinx para suporte aos seus processadores.

Por fim, é necessária a apresentação do mecanismo do Linux utilizado no sistema MicroBlaze para que o SO descubra quais os dispositivos inclusos na configuração do FPGA. O SO necessita desta informação para saber quais drivers carregar para tornar o sistema funcional.

Na arquitetura MicroBlaze, é utilizada uma descrição no formato FDT (Flattened Device Tree) que contém a descrição dos periféricos presentes no sistema. Este formato é utilizado também por diversas outras arquiteturas em Linux embarcado, como ARM, PowerPC e MIPS. Um sistema que utiliza FDT tem a descrição dos periféricos do sistema em um arquivo DTS (Device Tree Source), que é compilado no formato binário DTB (Device Tree Blob), e repassado para o Linux no momento da inicialização do sistema. A Listagem 4.1 mostra o trecho de um arquivo DTS.

O arquivo DTS deve ser descrito e compilado manualmente, ou podem ser utiliza-

Listagem 4.1 – Trecho do DTS de um sistema MicroBlaze

```
/dts-v1/;
/ {
    compatible = "xlnx,microblaze";
    /* ... */
    cpus {
        microblaze_0: cpu@0 {
            bus-handle = <&axi4lite_0>;
            clock-frequency = <100000000>;
            d-cache-baseaddr = <0xc0000000>;
            d-cache-highaddr = <0xdfffffff>;
            d-cache-line-size = <0x10>;
            d-cache-size = <0x4000>;
            /* ... */
            xlnx,use-fpu = <0x1>;
            xlnx,use-hw-mul = <0x2>;
            xlnx,use-icache = <0x1>;
            xlnx,use-interrupt = <0x1>;
            xlnx,use-mmu = <0x3>;
        } ;
    } ;
    ddr3_sdram: memory@c0000000 {
        device_type = "memory";
        reg = <0xc0000000 0x20000000>;
    } ;
    axi4lite_0: axi@0 {
        /* ... */
        crop_imgproc_master_0: crop-imgproc-master@71c00000 {
            compatible = "xlnx,crop-imgproc-master-1.00.a";
            interrupt-parent = <&microblaze_0_intc>;
            interrupts = <0 2>;
            reg = <0x71c00000 0x10000>;
            /* ... */
        } ;
        /* ... */
        microblaze_0_intc: interrupt-controller@41200000 {
            #interrupt-cells = <0x2>;
            compatible = "xlnx,axi-intc-1.02.a", "xlnx,xps-intc-1.00.a"
                ;
            interrupt-controller ;
            reg = <0x41200000 0x10000>;
            xlnx,kind-of-intr = <0xc>;
            xlnx,num-intr-inputs = <0x4>;
        } ;
        /* ... */
    } ;
} ;
```

das ferramentas para auxílio neste processo. O Xilinx EDK inclui um gerador automatizado de arquivos DTS de acordo com a configuração do sistema MicroBlaze.

O Linux realiza a leitura do DTB durante a inicialização, e assim instancia os dispositivos e drivers correspondentes aos periféricos existentes. Desta forma, é possível se utilizar um sistema de arquivos único que atenda a um conjunto de variações da plata-

forma, desde que inclua os drivers correspondentes às variações que precisa suportar.

Um dos atributos notáveis mostrados na Listagem 4.1 são a string `compatible`, que é utilizada pelo Linux para encontrar drivers compatíveis com aquele dispositivo. Os valores no atributo `reg` determinam endereço base e tamanho de áreas de mapeadas em memória para acesso aos periféricos. Com estas informações, os drivers podem ser implementados acessando endereços de memória relativos a `reg`. Assim, o Linux pode criar diversas instâncias do mesmo driver e é capaz de suportar diversas instâncias do mesmo dispositivo com um só driver.

4.3.2 Drivers Linux

Esta seção descreve o padrão proposto para implementação de drivers no Linux para possibilitar a implementação de aplicações na camada de Agentes que interagem com os periféricos customizados. O padrão é direcionado a aplicações que precisam transmitir maiores quantidades de dados a serem processados no periférico, visto que estes são os casos que tomam maior vantagem da plataforma. Este driver é implementado utilizando a abstração de manipulação de arquivos discutida na Seção 2.4.2.

A Figura 4.6 ilustra as operações que podem ser solicitadas ao driver pela aplicação do Agente para que possa solicitar uma operação de processamento na lógica programável.

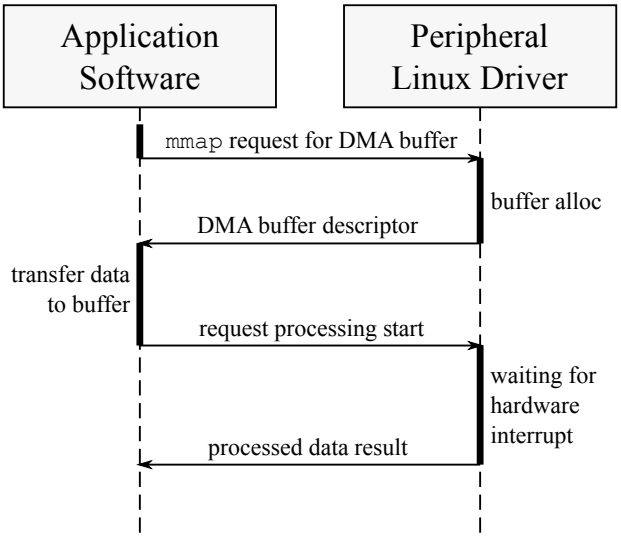
A primeira etapa a ser realizada pela aplicação é obter um buffer de memória compatível para transferência para dispositivo. É necessário que a aplicação utilize um buffer obtido via `mmap` pois este precisa ser alocado pelo kernel para que seja compatível com o uso pelo periférico. Requisitos que podem ser necessários para este buffer é que ele seja alocado em memória passível de DMA ou que seja contíguo em memória física.

A aplicação precisa então carregar os dados a serem processados para ser processada no buffer obtido.

A aplicação deve então requisitar para o driver que inicie o processamento no hardware. O driver pode implementar esta requisição com uma chamada de sistema `ioctl`.

Neste driver, o mesmo buffer de DMA utilizado para a passar os dados para o periférico pode ser utilizado como buffer de saída pelo periférico. Como este buffer pode estar alocado em memória fisicamente contígua, que é um recurso geralmente caro e limitado, este modelo é vantajoso pois requer somente a alocação de um buffer. Caso seja necessário processar ou descomprimir os dados pelo Agente, o buffer obtido via `mmap`

Figura 4.6 – Operações feitas pela aplicação no driver para requisitar processamento na lógica programável



Fonte: Autor

pode ser utilizado diretamente pelas a biblioteca de processamento e descompressão de dados, para evitar uma cópia adicional dos dados na aplicação..

Se o periférico suportar somente buffers de memória contígua para DMA, grandes buffers de DMA podem ser requisitados para que os dados seja armazenada por completo. Versões atualizadas do kernel Linux, incluindo a versão utilizada neste trabalho, proveem uma funcionalidade chamada CMA (Contiguous Memory Allocator) para a alocação deste tipo de buffer. A funcionalidade de CMA consiste de reservar um buffer de memória contíguo fisicamente durante a inicialização do sistema. Com o uso de CMA, quando um driver requisita ao SO um buffer de DMA, o buffer é alocado na memória contígua reservada. Esta reserva precisa ser feita durante a inicialização pois este é o único momento onde um grande buffer fisicamente contíguo pode existir: após a inicialização, a memória física pode se tornar rapidamente fragmentada. A fragmentação de memória física não é um problema particularmente grave para uma grande maioria de aplicações, sendo que o principal uso de CMA é para o uso com periféricos de hardware que requerem buffers fisicamente contíguos em memória para realizar DMA. Mesmo com o uso de CMA, deve ser considerado que os buffers alocados desta forma serão reservados apenas para esta operação, não ficando disponíveis para o resto do sistema enquanto o driver estiver carregado. Em sistemas embarcados, ao contrário de sistemas de propósito geral, esta desvantagem é amenizada pelo fato de que a quantidade de memória existente já foi provavelmente dimensionada para atender os requisitos da aplicação. Caso o hardware

implemente a funcionalidade mais complexa de *scatter/gather*, é eliminada a necessidade de se reservar um grande espaço para buffer DMA contíguo.

O driver deve disponibilizar uma forma de informar a aplicação sobre quando o processamento é concluído. Isto pode ser implementado no driver através de *polling* ou interrupção. Para uma implementação com *polling*, o driver pode implementar uma interface que retorna se o processamento já foi concluído. Esta forma é claramente menos eficiente pois requer verificação periódica do estado do processamento, o que faz uso desnecessário do processador.

Caso o hardware implemente um sinal de interrupção, uma forma mais eficiente é que o driver implemente uma chamada de sistema bloqueante, que retorne à aplicação quando a interrupção é retornada do periférico indicando que o processamento foi finalizado. Um exemplo de operação bloqueante é uma implementação de `read` que põe o processo em estado bloqueado pelo SO até que a operação seja concluída no hardware. Quando o driver retorna que a operação foi concluída, a aplicação pode acessar os dados processada através do buffer disponibilizado para o processamento.

Após acessar os dados processada, a aplicação pode desmapear o buffer (`munmap`) e terminar sua interação com o driver.

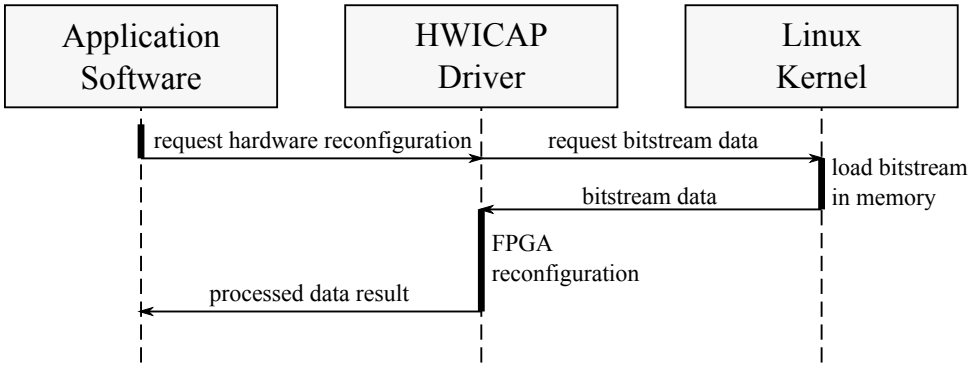
4.3.3 Driver HWICAP

Um dos objetivos da plataforma é permitir o uso de reconfiguração parcial de lógica programável para que seja possível alterar dinamicamente um algoritmo em hardware. A Xilinx disponibiliza um um IP Core em seus FPGAs, chamado HWICAP, que é capaz de realizar esta função.

Deve haver um método acessível por todo o sistema que seja capaz de reconfigurar partes da lógica programável através de um *bitstream*. Este método deve tomar conta de transmitir o *bitstream* para o HWICAP e efetuar a reconfiguração. Isso elimina a necessidade de que as aplicações implementem esta lógica, permitindo a reconfiguração dado somente o caminho para o *bitstream* desejado. Para esta plataforma, esta funcionalidade é implementada em um driver chamado driver HWICAP.

A figura Figura 4.7 mostra o funcionamento esperado do driver HWICAP. A aplicação é capaz de solicitar a reconfiguração apenas com o nome do *bitstream* desejado. Cada *bitstream* parcial já possui a informação da região reconfigurável onde será programado. O driver HWICAP solicita ao Linux para que carregue os dados em memória

Figura 4.7 – Operações feitas pela aplicação no driver HWICAP para requisitar reconfiguração parcial



Fonte: Autor

para que seja possível transmitir ao FPGA. Com o *bitstream* carregado em um buffer em memória, o HWICAP pode iniciar a reconfiguração escrevendo os dados no FPGA. Após terminar a reconfiguração, o driver HWICAP informa à aplicação que a operação foi finalizada.

Existem algumas observações quanto ao uso de reconfiguração parcial. A primeira observação é que as regiões de reconfiguração parcial são diferenciadas por região física do dispositivo, e não por algoritmo. Ou seja, caso se queira prover a opção de um conjunto de algoritmos em mais de uma região de reconfiguração parcial, deverá ser gerado um novo *bitstream* parcial por algoritmo específico para cada região física de reconfiguração parcial. O número de *bitstream*s necessário é então o número de algoritmos multiplicado pelo número de regiões de reconfiguração parcial.

O tamanho e tempo de reconfiguração das regiões de reconfiguração parcial dependem somente do tamanho da região de reconfiguração parcial reservada. Ou seja, mesmo que os algoritmos sejam mais simples, isso não reduz o tamanho ou tempo de reconfiguração de um *bitstream* parcial.

Uma outra observação sobre o uso de *bitstream* parciais é relacionado ao tempo de projeto. As ferramentas modernas da Xilinx possibilitam que a implementação da região estática do dispositivo (não-parcial) seja feita apenas uma vez. Portanto, para se definir novas possíveis implementações para uma região reservada de reconfiguração parcial, é necessário apenas reimplementar o bloco de reconfiguração parcial.

O tempo de reconfiguração utilizando *bitstream*s parciais pode ser significativamente menor do que o tempo de reconfiguração total do dispositivo. Isto pode ser considerado como uma vantagem adicional de se utilizar reconfiguração parcial nesta plataforma.

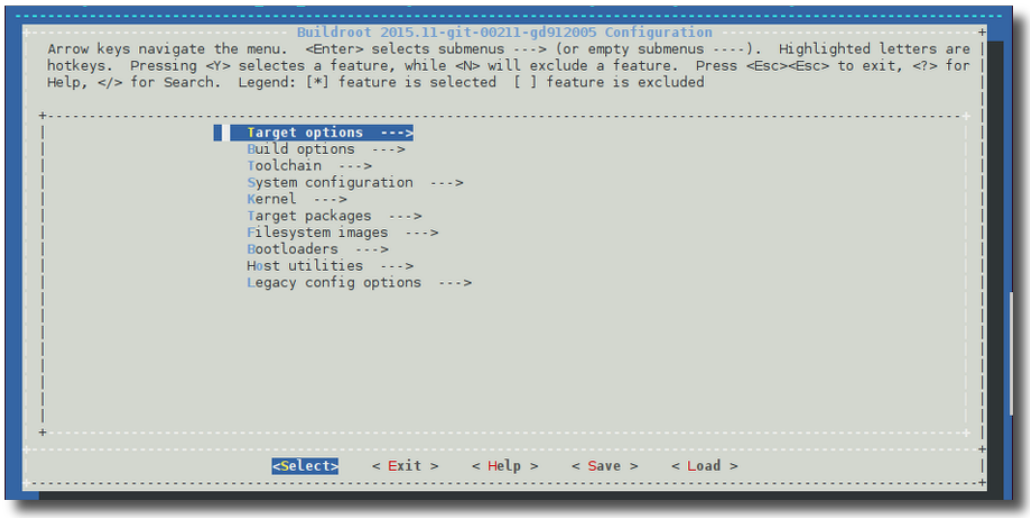
Este tempo é avaliado no Capítulo 5 utilizando *bitstreams* parciais.

4.3.4 Buildroot

O Buildroot é uma ferramenta simples, eficiente e simples de utilizar para se gerar sistemas com Linux embarcado através de cross-compilação (BUILDROOT, 2018).

Com o uso do Buildroot, é possível se gerar um sistema Linux com o mínimo para uma aplicação específica. É possível se gerar este sistema para uma variedade de arquiteturas de processador utilizadas em sistemas embarcados, incluindo MicroBlaze. O Buildroot orquestra a construção dos componentes básicos do sistema, como o compilador e conjunto de ferramentas de compilação para a arquitetura alvo (denominado de *toolchain* de cross-compilação), a construção do kernel Linux, DTB e do sistema de arquivos raiz. Através do menu apresentado na Figura 4.8, é possível se configurar as características do sistema gerado, que vão de escolha de arquitetura, escolha de tipos e nível de otimizações na compilação, até a escolha os pacotes de software que podem ser incluídos no sistema final. Dentre estes pacotes de software, é possível se escolher entre grande parte dos pacotes de software de código aberto existentes e que fazem parte de distribuições Linux comuns. Na versão 2015.11.1, que é a versão de referência utilizada no desenvolvimento desta plataforma, o Buildroot suporta 16 arquiteturas diferentes (além de variações) e aproximadamente 1400 pacotes de software.

Figura 4.8 – Captura de tela do `menuconfig` do Buildroot

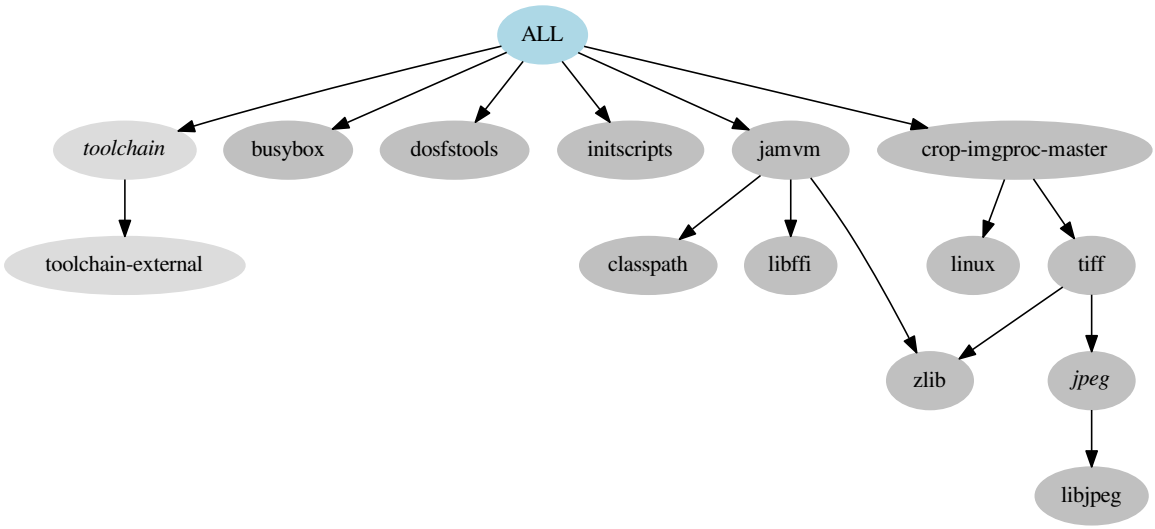


Fonte: buildroot.org (BUILDROOT, 2018)

Esta foi a ferramenta escolhida para esta plataforma pois ela possibilita a geração de um sistema básico minimalista contendo apenas as bibliotecas necessárias para a aplicação. Para suportar a plataforma, são utilizados principalmente os pacotes para instalação da JamVM, bibliotecas de carregamento de imagem de formatos como `tiff` e `jpeg`, compilação e instalação do kernel Linux. Estes pacotes formam um sistema básica, que pode ser estendido com os pacotes da aplicação do usuário.

A ferramenta é facilmente extensível, de forma que é possível incluir pacotes customizados para compilar o software da aplicação do usuário e incluí-lo junto no sistema final. Essa funcionalidade é utilizada para que o Buildroot também faça a compilação dos drivers, código JADE dos Agentes e aplicações de suporte necessárias.

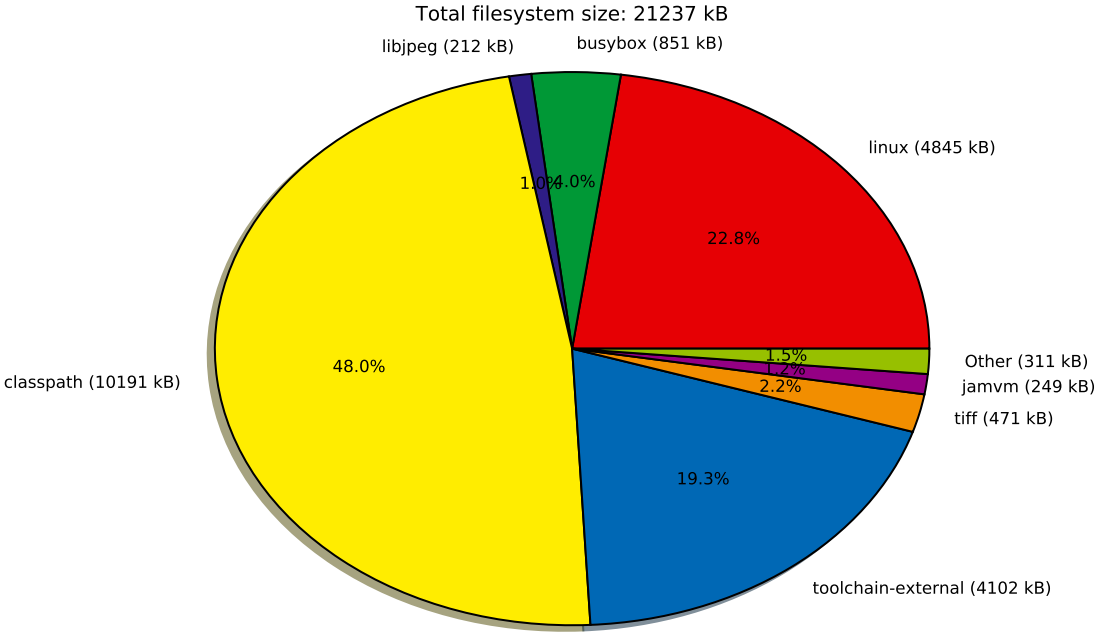
Figura 4.9 – Lista de pacotes de software de um sistema Linux embarcado base, gerado pelo Buildroot, e suas dependências



Fonte: Autor

A Figura 4.9 mostra os pacotes presentes no sistema base, além de sua relação de dependências. Ao ser invocado, o Buildroot faz a compilação cruzada (gerando binários para a arquitetura MicroBlaze) dos pacotes listados, respeitando as dependências existentes para que seja possível compilar com sucesso. O pacote `toolchain-external` representa, no sistema base, partes do conjunto de ferramentas de cross-compilação que são copiadas para o sistema base, como a biblioteca padrão C para a arquitetura MicroBlaze, dentre outras bibliotecas essenciais para a execução de programas C. O pacote `crop-imgproc-master` representa um pacote de usuário, que é o pacote desenvolvido para o estudo de caso apresentado na Seção 5.4. Este pacote foi incluído no Buildroot utilizando os métodos de extensão existentes na ferramenta.

Figura 4.10 – Tamanho de cada pacote de software de um sistema Linux embarcado base, gerado pelo Buildroot, e a respectiva porcentagem no tamanho final do sistema



Fonte: Autor

Uma das vantagens do uso do Buildroot com relação a outros sistemas de build embarcados ou distribuições Linux embarcadas é a possibilidade de se gerar imagens com tamanho reduzidas, otimizadas para a aplicação. A Figura 4.10 apresenta o tamanho do sistema final e a contribuição de cada pacote de software no tamanho deste sistema final. O espaço necessário é da ordem de 20MB, incluindo as bibliotecas para execução do Java, o kernel Linux, e uma aplicação (`crop-imgproc-master`). É possível observar que as bibliotecas para execução do Java (instaladas pelo pacote `classpath`) representam quase metade do sistema final, mostrando que o Java é um ambiente relativamente pesado para este tipo de sistema. Apesar de o espaço de armazenamento não ser mais um dos fatores limitantes em muitos sistemas embarcados, o modesto tamanho deste sistema reforça o uso eficiente dos recursos para sistema MicroBlaze.

4.3.5 Agentes JADE

O framework de Agentes JADE, apresentado na Seção 2.2, oferece a possibilidade de migração de Agentes entre plataformas do framework. Este recurso pode ser explo-

rado em conjunto com a reconfiguração parcial para que sejam disponibilizados múltiplos Agentes reconfiguráveis em hardware. Por exemplo, a funcionalidade de migração de Agentes do JADE pode ser utilizada de forma que sempre que um Agente migra para uma plataforma que possui recursos de lógica programável livres e tem disponível uma implementação em hardware para ser utilizada, o Agente reconfigura a região livre para implementar seu algoritmo.

4.3.6 JamVM

Para se executar o JADE 4.4.0, é necessário o uso de uma JVM com suporte à especificação 1.5 ou maior.

A JamVM é uma JVM de código aberto que tem por objetivo suportar a última versão das especificações para JVMs, ao mesmo tempo em que se mantém compacta e fácil de entender (LOUGHER, 2014).

Nesta plataforma, é utilizada a JamVM pois a mesma se mostrou facilmente portável para arquiteturas como o MicroBlaze.

A versão 2.0.0 da JamVM é utilizada como referência nesta implementação, com patches para funcionamento na arquitetura MicroBlaze.

A JamVM provê a implementação da máquina virtual, porém depende de uma implementação de biblioteca padrão de classes Java externa. A JamVM é compatível com as implementações *GNU Classpath* ou a biblioteca de classes do *OpenJDK*.

Por padrão, a JamVM é configurada para utilizar a biblioteca *GNU Classpath*, devido ao fato de ser menor e portanto mais indicada para sistemas embarcados (Free Software Foundation, Inc., 2018).

4.4 Migração de Agentes JADE com reconfiguração parcial

Esta seção apresenta uma proposta de Agentes auxiliares para a implementação de reconfiguração parcial na plataforma.

Para que seja possível realizar o uso de reconfiguração parcial em conjunto com o JADE, é necessário que haja uma interface entre os Agentes e a camada de reconfiguração parcial do hardware. É importante que esta camada seja capaz de abstrair a existência e a real implementação da reconfiguração do hardware, para que ainda seja possível realizar

a migração de agentes livremente pelo SMA.

Para isso são propostas duas classes de Agentes para implementação do experimento, *HwReconfAgent* e *HwReconfRequesterAgent*. A plataforma faz também uso de Agentes padrão do JADE, como o AMS e o DF, introduzidos na Seção 2.2.

A classe HwReconfAgent é responsável por gerenciar as regiões reconfiguráveis do container onde está executando. Um Agente desta classe conhece as regiões reconfiguráveis disponíveis e as possibilidades existentes para reconfiguração de cada região. Na plataforma proposta, isto se traduz em conhecer as regiões de reconfiguração parcial e os *bitstreams* disponíveis para cada região. Durante sua inicialização, um Agente da classe HwReconfAgent registra um serviço chamado `hwreconf` no DF indicando que é capaz de atender requisições de reconfiguração de hardware. Assim, cada novo Agente pode verificar a existência deste serviço no DF para a possibilidade de reconfiguração de hardware no container onde está inserido.

A classe HwReconfAgent utiliza os serviços implementados no driver HWICAP apresentado na Seção 4.3.3 para realizar a real reconfiguração no hardware. Isto também elimina a necessidade de implementação da rotina de reconfiguração parcial em cada novo Agente da plataforma. Uma responsabilidade importante do HwReconfAgent é de manter a lista de regiões reconfiguráveis que estão sendo utilizadas, de forma que não haja conflitos de utilização e que uma região não seja reconfigurada enquanto estiver sendo utilizada por um Agente.

O HwReconfAgent é configurável a partir de um arquivo `xml` `hwreconf.xml` como o apresentado na Listagem 4.2. O arquivo `hwreconf.xml` descreve as regiões de reconfiguração parcial disponíveis e os seus *bitstreams* associados. O formato `xml` é utilizado para facilitar o carregamento do arquivo com bibliotecas Java existentes. Conforme mencionado na Seção 4.3.3, cada implementação de um algoritmo em hardware precisa de um bitstream especificamente gerado para cada região reconfigurável, portanto é necessário descrever neste arquivo quais as opções para cada região.

Após ser instanciado e carregar o arquivo de configuração, um Agente HwReconfAgent fica escutando por requisições do tipo PROPOSE, vindo de Agentes que querem realizar reconfiguração no hardware. Ao receber uma requisição, o HwReconfAgent verifica se a requisição é uma das disponíveis na sua configuração. Caso seja uma das opções disponíveis, realiza a reconfiguração no hardware e retorna uma mensagem ACCEPT-PROPOSAL. Caso não seja uma das opções disponíveis, retorna uma mensagem REJECT-PROPOSAL. Por fim, o HwReconfAgent aceita também mensagens CAN-

Listagem 4.2 – Exemplo de `hwreconf.xml` para o Agente HwReconfAgent

```
<?xml version="1.0"?>
<hwreconf>
    <region>
        <id>1</id>
        <service>
            <name>imgproc(256x256)</name>
            <bitstream>imgproc256.bit</bitstream>
        </service>
        <service>
            <name>imgproc(512x512)</name>
            <bitstream>imgproc512.bit</bitstream>
        </service>
    </region>
    <region>
        <id>2</id>
        <service>
            <name>mult</name>
            <bitstream>mult.bit</bitstream>
        </service>
        <service>
            <name>add</name>
            <bitstream>add.bit</bitstream>
        </service>
    </region>
</hwreconf>
```

CEL para marcar uma região reconfigurável como não disponível.

A segunda classe de Agentes, HwReconfRequesterAgent, é implementada para validação em estudos de caso. Um Agente desta classe realiza a consulta pelo serviço `hwreconf` no DF ao ser instanciado ou migrado para outro container. Sendo assim, ele pode ser migrado entre os containeres para testar a comunicação e funcionamento do Agente HwReconfAgent.

A Figura 4.11 mostra um exemplo com migrações de um Agente HwReconfRequesterAgent entre diferentes plataformas. Na figura, é utilizado o Agente *Sniffer* do JADE para monitoramento das mensagens de migração com reconfiguração parcial.

O exemplo considera três containeres e os seguintes Agentes (são omitidos os Agentes padrão do tipo DF, AMS, *Sniffer* apresentados na Seção 2.2):

- `think`: container executando em uma plataforma Desktop. Neste exemplo, é utilizado apenas para monitoramento do SMA e visualização do Agente *Sniffer*.
- `mini`: container executando na plataforma BeagleBone, que não possui recursos de lógica programável, portanto pode apenas executar Agentes em software. O Agente *hwrmini* é do tipo HwReconfAgent, e responde as requisições de reconfiguração parcial neste container. Como esta plataforma não possui hardware reconfigurável,

o Agente serve para testar a resposta de requisições em casos onde não é possível.

- `agentmb`: container executando na plataforma que possui capacidades de reconfiguração parcial de hardware. O Agente *hwragentmb* é do tipo HwReconfAgent, e responde as requisições de reconfiguração parcial neste container. Neste exemplo, este é o Agente que é capaz de atender as requisições e reconfigurar o hardware.

Na Figura 4.11, é possível ver as seguintes sequências de eventos relacionados ao exemplo.

0-7. Criação do Agente service1 no container `agentmb`.

8-9. Solicitação e informação sobre serviços locais capazes de realizar reconfiguração em hardware do FPGA. O Agente DF responde indicando que o Agente `hwragentmb` oferece este serviço.

10-11. Pedido de reconfiguração em hardware do FPGA feita pelo Agente service1 para o `hwragentmb` (PROPOSE), e a confirmação de que o pedido foi atendido (ACCEPT-PROPOSAL).

12-14. Solicitação da migração do Agente service1 do container `agentmb` para o container `mini`.

15-16. Solicitação de liberação do recurso de hardware que o Agente service1 estava utilizando.

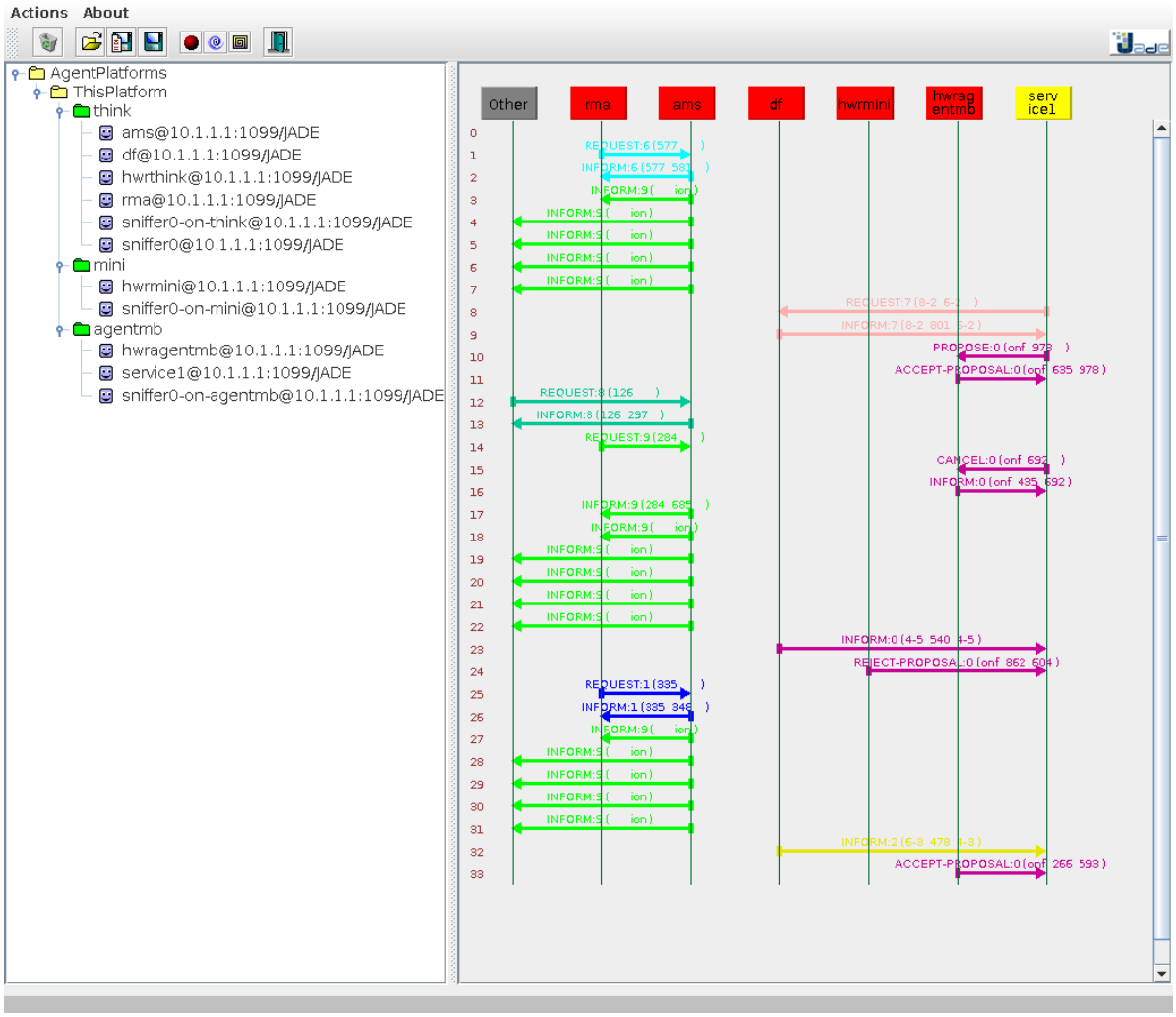
17-22. Migração do Agente service1 do container `agentmb` para o container `mini`.

23. Informação do Agente DF sobre serviços de reconfiguração em hardware no container `mini`.

24. Resposta do Agente `hwrmini` indicando que não tem recursos de reconfiguração em hardware disponíveis para implementação do Agente service1 em hardware (REJECT-PROPOSAL).

25-33. Migração do Agente service1 do container `mini` de volta para o container `agentmb`. O processo é semelhante aos itens anteriores, mas é possível ver que ao retornar para o container `agentmb`, o Agente `hwragentmb` informa que é capaz de implementar o serviço em hardware (ACCEPT-PROPOSAL).

Figura 4.11 – Monitoramento de migração de Agentes e reconfiguração de Agentes em hardware utilizando o Agente *Sniffer*



Fonte: Autor

4.5 Considerações finais

Nesta seção foram apresentadas a arquitetura da plataforma e a descrição da implementação para suporte de Agentes com reconfiguração parcial. A arquitetura proposta é composta por camadas de hardware, hardware reconfigurável, software de plataforma a nível de SO, e software de alto nível. Por requerer o projeto de hardware e software, a implementação de uma plataforma que atende a arquitetura proposta não é trivial. No entanto, a plataforma resultante é bastante flexível e graças ao uso de Agentes através do framework JADE pode atender requisitos como flexibilidade, robustez e reconfigurabilidade.

As principais contribuições da plataforma com relação aos trabalhos existentes são:

- Utilização do processador soft core reduz os requisitos de hardware por não necessitar de um processador externo para executar o framework de Agentes.
- Suporte a periféricos no estilo DMA possibilita melhorias de desempenho para aplicações que requerem processamento de maiores volumes de dados em periféricos de hardware.
- Suporte a reconfiguração parcial possibilita maior flexibilidade para implementação de Agentes em hardware.

Os estudos de caso apresentados no Capítulo 5 buscam avaliar a efetividade da plataforma ao atingir estes objetivos.

5 ESTUDOS DE CASO

Neste capítulo, são apresentados os experimentos e estudos de caso utilizados para validar este trabalho. Os resultados são apresentados após a descrição de cada estudo de caso. Além de propor os estudos de caso para validação, o detalhamento das possibilidades de implementação dos estudos de caso também é apresentado neste capítulo.

Inicialmente, as plataformas embarcadas utilizadas nos experimentos são apresentadas na Seção 5.1.

Uma das vantagens apontadas pela arquitetura apresentada é a capacidade de processamento de grandes quantidades de dados no FPGA, portanto alguns dos experimentos são propostos visando medir a capacidade de ganhos em aplicações neste contexto, como processamento de sinais e imagem. Na Seção 5.2 e na Seção 5.3 são apresentados estudos de caso nesta área.

Na Seção 5.3.3 é apresentado um estudo de caso mais complexo envolvendo uma potencial aplicação prática da plataforma, em uma aplicação em agricultura de precisão utilizando VANTs com processamento de imagens.

Por fim, são apresentados alguns experimentos adicionais realizados durante o desenvolvimento do trabalho na Seção 5.6.

5.1 Plataformas embarcadas utilizadas

Ao todo, foram utilizados cinco plataformas embarcadas no decorrer dos experimentos realizados pelo trabalho. Com exceção da plataforma que contém a implementação de referência deste trabalho, nem todos os experimentos compartilham o mesmo conjunto de plataformas embarcadas, pois os experimentos foram realizados em contextos diferentes durante o período de desenvolvimento. Na Tabela 5.1 é apresentado um resumo das plataformas embarcadas utilizadas, e nas subseções a seguir são apresentados mais detalhes de cada uma delas.

5.1.1 ML605

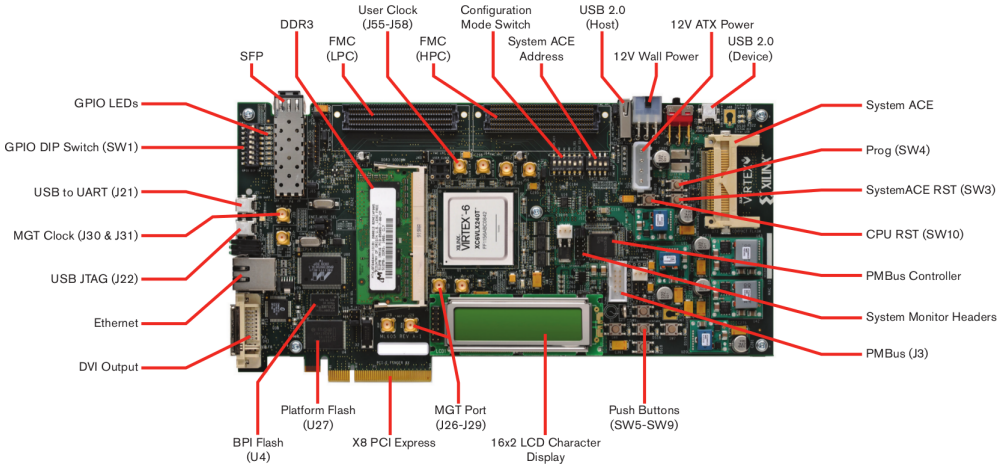
A plataforma de desenvolvimento Xilinx ML605 foi utilizada como referência para a implementação da arquitetura descrita. A plataforma é representada na Figura 5.1.

Tabela 5.1 – Resumo das plataformas utilizadas nos estudos de caso

Plataforma	Arquitetura	Frequência	Cores	Memória	SIMD	FPGA
Raspberry Pi	armv6	700 MHz	1	512 MB	NEON	–
BeagleBone	armv7	720 MHz	1	256 MB	NEON	–
Cubieboard2	armv7	1 GHz	2	1 GB	NEON	–
Desktop	x86_64	2.5 GHz	4	10 GB	SSE	–
ML605	microblaze	100 MHz	1	512 MB	–	Virtex-6

Fonte: Autor

Figura 5.1 – Recursos da plataforma de desenvolvimento Xilinx ML605



Fonte: (XILINX, 2012)

A plataforma ML605 utiliza o FPGA Virtex-6 XC6VLX240T. Os FPGAs da linha Virtex, da Xilinx, são voltados para atender projetos com requisitos de alto desempenho, projetos com demanda de processamento de sinais digitais, e projetos que precisam fazer uso de capacidades de processamento em software em conjunto com hardware, através do uso de processadores soft core (XILINX, 2015). Este alvo foi considerado adequado para o trabalho devido à possibilidade de utilização do processador soft core e devido às facilidades oferecidas pelas ferramentas da Xilinx para implementação de periféricos customizados para o processador.

Neste trabalho foi utilizado o processador MicroBlaze para fazer a interface entre a lógica programável e a lógica implementada em FPGA. O uso do processador MicroBlaze com possibilita a implementação de funcionalidades de Agentes em conjunto com a implementação de Agentes em hardware.

O MicroBlaze utilizado na implementação feita é da versão 8.40.a, executa à

frequência de 100MHz, e foi customizado através da ferramenta Xilinx EDK 14.2. O processador MicroBlaze se comunica com os periféricos no FPGA através do barramento AXI, que também executa a 100MHz. O MicroBlaze faz uso de memória externa 512MB de memória DDR3. O MicroBlaze também é capaz de utilizar outros hardware providos pela placa, como conector e PHY Ethernet 1000/100/10, além de 2GB de memória persistente através de um cartão *CompactFlash*. O processador MicroBlaze foi configurado para utilizar uma MMU, que o permite executar um SO completo como o Linux. Com o uso do Linux, é possível executar a JVM JamVM, que possibilita a execução do JADE.

O sistema Linux embarcado gerado através do Buildroot, conforme detalhado na Seção 4.3.4 Esta plataforma é referida nos resultados como plataforma ML605 ou apenas MicroBlaze, quando referido à execução em software.

5.1.2 Raspberry Pi

A SBC Raspberry Pi é um computador do tamanho de um cartão de crédito, que pode ser acoplada a uma câmera 5MP, que é conveniente para uso em conjunto com um VANT devido ao seu peso e dimensões reduzidas. Esta plataforma é apresentada como o módulo de controle de missão na Seção 5.4. O modelo utilizado nos estudos de caso foi o Raspberry Pi B+, que possui um processador ARM1176JZF-S (armv6), executando a 700MHz. Um recurso notável da Raspberry Pi para esta aplicação é que o processador provê a extensão ARM NEON, que possibilita a execução de instruções SIMD (Single instruction, multiple data). Através do uso destas instruções, é possível acelerar a execução de operações vetoriais, como processamento de imagem, em software. Apesar de possuir maior poder de processamento do que a plataforma ML605, esta plataforma não é capaz de executar Agentes em hardware. O SO executado pela Raspberry Pi é o Raspbian. Esta plataforma é referida nos resultados como plataforma Raspberry Pi.

5.1.3 BeagleBone

Em outros experimentos, é utilizada outra SBC com processador ARM, denominada BeagleBone. Esta plataforma possui um SOC Texas Instruments AM3359, que opera a 720MHz e possui um núcleo de processador ARM Cortex-A8 (armv7). Assim como a Raspberry Pi, o processador provê a extensão ARM NEON, que possibilita a exe-

cução de instruções SIMD. Nos experimentos onde esta plataforma é utilizada, ela executa um ambiente de software sistema Linux embarcado gerado através do Buildroot, muito semelhante ao utilizado pela ML605, descrito na Seção 4.3.4 O software utilizado é o mesmo sistema Buildroot utilizado para a plataforma ML605, porém gerado e otimizado para a arquitetura ARM ao invés de MicroBlaze. Esta plataforma não possui recursos de hardware reconfigurável, então somente a funcionalidade em software é compilada para a BeagleBone. Esta plataforma é referida nos resultados como plataforma BeagleBone.

5.1.4 Cubieboard2

Em alguns experimentos, é utilizada uma terceira SBC com processador ARM, denominada Cubieboard2. A plataforma possui um SOC Allwinner A20, que opera a 1GHz e possui dois núcleos de processador ARM Cortex-A7 (armv7). Assim como a Raspberry Pi, o processador provê a extensão ARM NEON, que possibilita a execução de instruções SIMD. A Cubieboard2 é utilizada como referência para um sistema embarcado com poder de processamento relativamente alto, e que é capaz de executar Agentes JADE. Esta plataforma possui o maior desempenho entre as plataformas de referência ARM. Apesar de possuir maior poder de processamento do que a plataforma ML605, esta plataforma não é capaz de executar Agentes em hardware. Esta plataforma é referida nos resultados como plataforma Cubieboard2.

5.1.5 Desktop

Para fins de referência e avaliação de desempenho, um computador Desktop convencional também é utilizado. O computador possui um processador x86_64 Intel i5-2450M a 2.50GHz, executando o SO Ubuntu 14.04. Esta plataforma possui o maior desempenho de software entre todas as plataformas utilizadas. Esta plataforma é referida nos resultados como plataforma Desktop.

5.2 Algoritmos de processamento de sinal

Algoritmos de processamento de sinal são uma aplicação típica que pode ser otimizada com implementação em FPGA. FPGAs são um bom alvo para esta aplicação

devido à possibilidade de implementar hardware especializado para realizar as diversas operações matemáticas necessárias em menos ciclos do que um processador de propósito geral que possui unidades aritméticas limitadas.

Algoritmos de processamento de sinal também são uma aplicação interessante para avaliação em FPGA pois é possível se realizar a implementação em hardware com o auxílio de ferramentas como o MATLAB®.

Por se tratar de um conjunto de operações de multiplicação e soma, implementações dos algoritmos de processamento de sinal baseadas em software também podem receber otimizações em processadores capazes de realizar operações vetoriais. Por isso, é importante avaliar também se é vantajoso investir nos recursos de lógica programável para otimizar este problema, em relação à maior facilidade de se utilizar unidades de processamento vetorial em software.

Nesta seção são apresentados os resultados de tempo de execução de alguns experimentos escolhidos na categoria de algoritmos de processamento de sinal, na plataforma ML605 e na plataforma BeagleBone. Os experimentos apresentados são o de um filtro FIR (Finite Impulse Response) simples de uma dimensão e um filtro de Kalman.

Estes experimentos foram escolhidos como estudo de caso pois são algoritmos típicos de implementação em FPGA. O objetivo é avaliar as possibilidades de ganho de desempenho com as implementações em hardware. Os experimentos foram executados separadamente em cada uma das plataformas. Nestes experimentos, o JADE não foi utilizado.

5.2.1 Filtro FIR

Os filtros de resposta ao impulso finito, ou filtros FIR, são os tipos mais simples de filtros digitais para se analisar. Os filtros FIR possuem este nome pois cada valor inserido no sinal de entrada tem uma contribuição *finita* no sinal de saída.

Um filtro pode ser representado matematicamente conforme representado pela Equação (5.1) (LYONS, 2004).

$$y[n] = \sum_{i=0}^{N} h_i \cdot x[n-i] \tag{5.1}$$

Onde:

- $x[n]$ é o sinal de entrada,

- $y[n]$ é o sinal de saída,
- N é a ordem do filtro,
- h_i é o i-ésimo coeficiente de resposta

Os filtros FIR em geral possuem implementação simples em dispositivos DSP (Digital Signal Processor) ou FPGA, onde, para filtros simples, é possível realizar uma iteração do filtro através de uma única instrução ou ciclo de clock. Os filtros FIR são amplamente empregados como parte de diversas aplicações, em funções como como supressão de ruídos, aprimoramento de sinais. Um exemplo de aplicação de filtro FIR é um filtro de média ou passa-baixa, que gera na sua saída um sinal com altas frequências filtradas.

Neste estudo de caso, é considerado um filtro FIR com 4 coeficientes, apresentados na Equação (5.2).

$$\begin{aligned} h_0 &= -0.1339 \\ h_1 &= -0.0838 \\ h_2 &= 0.2026 \\ h_3 &= 0.4064 \end{aligned} \tag{5.2}$$

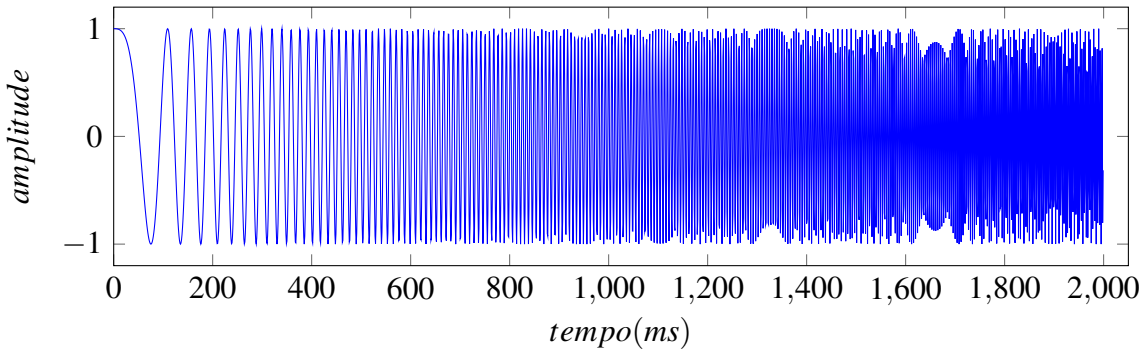
O objetivo deste filtro é de exemplificar uma operação de filtro de sinal. Os coeficientes utilizados foram obtidos no filtro apresentado em (MATHWORKS, 2015b). O algoritmo não é dependente deste número específico de coeficientes ou dos valores específicos.

A Figura 5.2 mostra uma representação de aplicação do filtro com os coeficientes descritos em um sinal de exemplo.

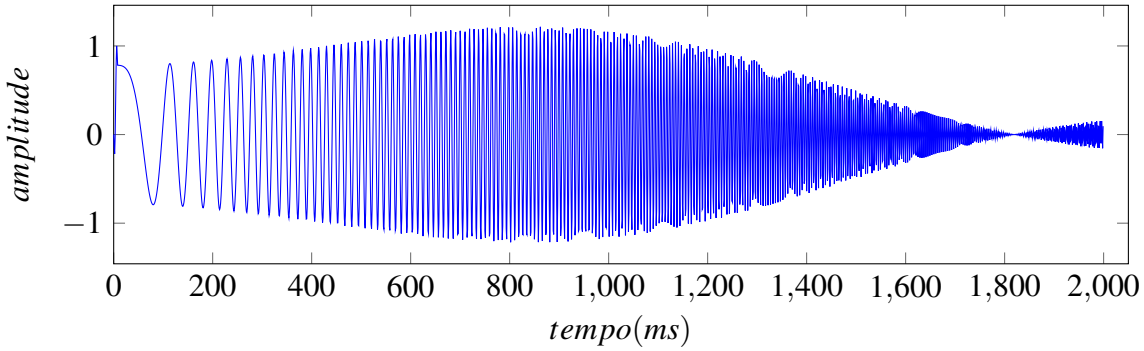
O filtro FIR é um problema amplamente estudado e pode ser otimizado de diversas formas. No contexto de otimização com hardware programável, o estudo de caso com filtro FIR tem por objetivo possibilitar comparações de diferentes formas de interação com o software. É possível implementar o filtro FIR em hardware de forma que o software envia uma amostra por vez para o processamento em hardware, ou de forma que o hardware processe uma carga maior de dados por vez. Através da implementação das duas formas, é possível avaliar melhor o compromisso entre as duas implementações.

O algoritmo base para este estudo de caso em específico é disponível em (MATHWORKS, 2015b). Dentre os algoritmos utilizados para os experimentos, o filtro FIR é o algoritmo que requer menor intensidade de poder computacional. Ambas as

Figura 5.2 – Execução do algoritmo Filtro FIR



(a) Sinal de entrada (com ruído)



(b) Sinal de saída (filtrado)

Fonte: Autor

implementações em hardware e software para este estudo de caso foram geradas automaticamente através da ferramenta MATLAB®.

Na plataforma ML605, o bloco de processamento do filtro FIR em hardware utiliza aritmética de ponto fixo, com números de 32 bits sendo 16 bits para a parte fracionária. Esta divisão de bits foi escolhida após a análise da ferramenta sobre o conjunto de dados a ser utilizado no experimento. O bloco de controle do filtro FIR é gerado utilizando a ferramenta Xilinx EDK.

O periférico de hardware para o filtro FIR é inserido no barramento AXI para prover acesso ao MicroBlaze. Foram implementados dois periféricos, um que provê acesso via MMIO e outro que provê acesso via DMA. A versão que provê acesso MMIO requer que o processador entregue uma amostra por acesso ao espaço de memória do periférico. A versão que provê acesso DMA realiza acessos de 16 amostras de dados a cada requisição de leitura à memória.

A versão que provê acesso MMIO é implementada para fins de comparação com (CEMIN; GOTZ; PEREIRA, 2012), cujos resultados são apresentados com acessos

MMIO.

A versão em software utilizada na ML605 e na BeagleBone implementa o mesmo algoritmo, utilizando aritmética em ponto flutuante de precisão simples (32 bits). Na BeagleBone, a versão em software é compilada fazendo uso das instruções SIMD disponíveis no processador.

5.2.2 Filtro de Kalman

O filtro de Kalman é um algoritmo que utiliza uma série de medidas contendo ruído e outras incertezas, e produz estimativas de variáveis desconhecidas através do uso de probabilidades com base nas medidas observadas. O filtro possui este nome devido à atribuição ao seu inventor em (KALMAN, 1960).

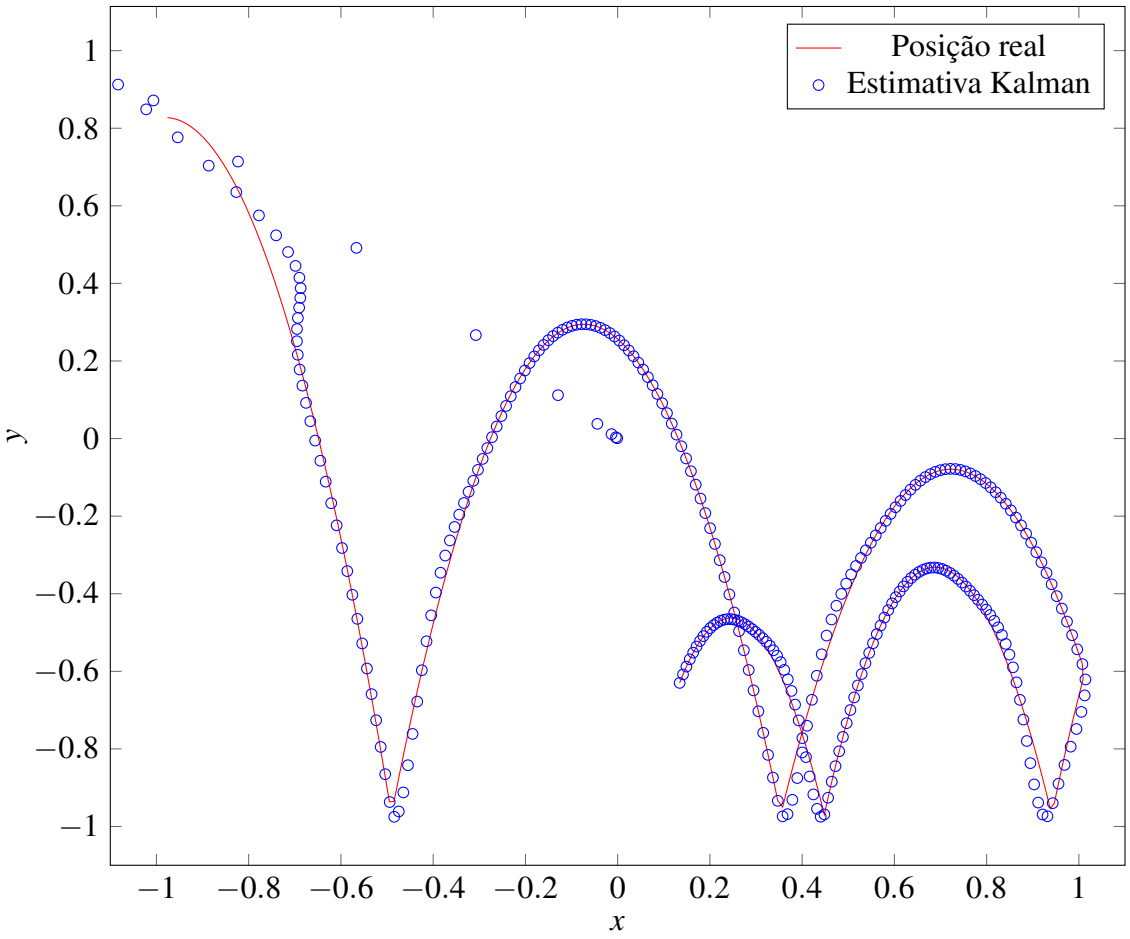
A Figura 5.3 mostra uma representação do sinal utilizado para testes com o filtro de Kalman. O exemplo na imagem representa um objeto que é solto a partir do ponto $(-1,1)$ inicialmente se movimentando em direção ao eixo positivo de x. Na figura, é possível ser observar que alguma estimativas de posição são dadas próximos do ponto $(0,0)$. Estas estimativas ocorrem no início da execução do algoritmo, enquanto ainda não existem amostras suficientes para se realizar uma boa estimativa. Após algumas amostras, o filtro consegue realizar uma melhor estimativa da posição real.

O algoritmo base para este estudo de caso em específico é disponível em (MATHWORKS, 2015a). Dentro os algoritmos utilizados, o filtro de Kalman é um dos que requer a maior quantidade de poder computacional e recursos de hardware. Ambas as implementações em hardware e software para este estudo de caso foram geradas automaticamente através da ferramenta MATLAB®.

Na plataforma ML605, o bloco de processamento do filtro de Kalman em hardware utiliza aritmética de ponto fixo, com números de 32 bits sendo 30 bits para a parte fracionária. Esta divisão de bits foi escolhida após a análise da ferramenta sobre o conjunto de dados a ser utilizado no experimento. O bloco de controle do filtro de Kalman é gerado utilizando a ferramenta Xilinx EDK.

O periférico de hardware para o filtro de Kalman é inserido no barramento AXI para prover acesso ao MicroBlaze. Para o algoritmo filtro de Kalman, não foi possível realizar a implementação com DMA devido à alta quantidade de recursos demandada pela saída da ferramenta MATLAB®, portanto somente a versão MMIO foi avaliada. Esta versão requer que o processador entregue uma amostra por acesso ao espaço de memória

Figura 5.3 – Execução do algoritmo Filtro de Kalman



Fonte: Autor

do periférico.

A versão em software utilizada na ML605 e na BeagleBone implementa o mesmo algoritmo, utilizando aritmética em ponto flutuante de precisão simples (32 bits). Na BeagleBone, a versão em software é compilada fazendo uso das instruções SIMD disponíveis no processador.

5.2.3 Resultados

A Tabela 5.2 mostra os resultados de tempo de execução das variações testadas para os algoritmos de processamento de sinal. Conforme mencionado na Seção 5.2.2, para o algoritmo filtro de Kalman, não foi possível realizar a implementação com DMA devido à alta quantidade de recursos demandada para a implementação. Para o algoritmo filtro FIR, é utilizada um sinal com 10000 amostras. Para o algoritmo filtro Kalman, é

utilizado um sinal com 300 amostras de pontos em um plano 2D. Para cada experimento, o algoritmo foi executado 100 vezes em cada variação, e o valor apresentado é a média dos tempos medidos.

Tabela 5.2 – Tempo de execução algoritmos Filtro FIR e Filtro de Kalman

	Tempo de execução (ms)			
		ML605		
Algoritmo	BeagleBone	Software	Hardware MMIO	Hardware DMA
fir	416	954	945	385
kalman	201	1495	729	—

Fonte: Autor

Nas implementações em software, cada amostra é processada em um software de aplicação no Linux. Na implementação em hardware MMIO, cada amostra é processada em hardware, porém a aplicação faz uma chamada de sistema para processar cada amostra em hardware. Na implementação em hardware DMA, toda a janela de amostras é carregada no buffer DMA e repassada para processamento em hardware. O hardware gera uma interrupção quando todas as amostras são processadas e o buffer utilizado no DMA já contém o resultado final.

É notável que a implementação com MMIO pode ser mais lenta do que uma implementação otimizada em software e isto pode ser atribuído ao custo das chamadas de sistema extras que são feitas para cada amostra. Por isso, nestas aplicações, o ganho observado não é significativo. A quantidade de dados a processar também não é extremamente elevada, o que acentua o *overhead* das operações em software como as chamadas de sistema.

O uso de DMA reverte este quadro mesmo para o processador MicroBlaze no filtro FIR. O esperado é que o ganho se torne mais significativo conforme a quantidade de dados se torna maior.

5.3 Algoritmos de processamento de imagem

Algoritmos de processamento de imagem são uma aplicação típica que pode ser otimizada com implementação em FPGA. FPGAs são um bom alvo para esta aplicação devido à possibilidade de implementar hardware especializado para realizar as diversas

operações matemáticas necessárias em menos ciclos do que um processador de propósito geral que possui unidades aritméticas limitadas.

Algoritmos de processamento de imagem também são uma aplicação interessante para avaliação em FPGA pois é possível se realizar a implementação em hardware com o auxílio de ferramentas como o MATLAB®.

Por se tratar de um conjunto de operações como multiplicações de matrizes, que podem se beneficiar da disponibilidade de multiplicadores em paralelo em hardware, implementações dos algoritmos de processamento de imagem baseadas em software também podem receber otimizações em processadores capazes de realizar operações vetoriais. Por isso, é importante avaliar também se é vantajoso investir nos recursos de lógica programável para otimizar este problema, em relação à maior facilidade de se utilizar unidades de processamento vetorial em software.

Nesta seção são apresentados os resultados de tempo de execução de alguns experimentos escolhidos na categoria de algoritmos de processamento de imagem, na plataforma ML605 e na plataforma BeagleBone. Os experimentos apresentados são o de um filtro 2D para processamento de imagem (algoritmo de redução de ruído) e um algoritmo de conversão de espaço de cores (*rgb2yuv*).

Estes experimentos foram escolhidos como estudo de caso pois são algoritmos típicos de implementação em FPGA. O objetivo é avaliar as possibilidades de ganho de desempenho com as implementações em hardware. Os experimentos foram executados separadamente em cada uma das plataformas. Nestes experimentos, o JADE não foi utilizado.

5.3.1 Redução de ruído

O algoritmo de redução de ruído de imagens consiste de um filtro de imagem de convolução. Um filtro de imagem é uma transformação da imagem pixel a pixel, é calculado com base no nível de cor do pixel levando em conta os níveis de cor dos pixels adjacentes. O processo de filtragem é feito utilizando matrizes, neste algoritmo também denominadas máscaras, as quais são aplicadas sobre a imagem através de convolução de matrizes.

Na Equação (5.3), é apresentada a matriz de coeficientes 3×3 que pode ser utili-

zada para redução de ruído:

$$mask = \begin{pmatrix} -0.1667 & -0.6667 & -0.1667 \\ -0.6667 & 4.3333 & -0.6667 \\ -0.1667 & -0.6667 & -0.1667 \end{pmatrix} \tag{5.3}$$

Dependendo da máscara utilizada, é possível se obter diversos tipos de filtros. É possível se projetar filtros para realçar parâmetros da imagem, como bordas, ou determinadas "direções" da imagem.

Imagens onde o objeto alvo está borrado devido a ruídos no sensor ou obtidas com câmera fora de foco podem ser tratadas por filtros de redução de ruído para melhorar os resultados do processamento de imagem.

A Figura 5.4 mostra uma representação do sinal utilizado para testes com o algoritmo de redução de ruído.

O algoritmo base para este estudo de caso em específico é disponível em (MATHWORKS, 2015d). Ambas as implementações em hardware e software para este estudo de caso foram geradas automaticamente através da ferramenta MATLAB®.

Na plataforma ML605, o bloco de processamento do filtro de redução de ruído em hardware utiliza aritmética de ponto fixo, com números de 32 bits sendo 15 bits para a parte fracionária. Esta divisão de bits foi escolhida após a análise da ferramenta sobre o conjunto de dados a ser utilizado no experimento. O bloco de controle do filtro de redução de ruído é gerado utilizando a ferramenta Xilinx EDK.

O periférico de hardware para o filtro de redução de ruído é inserido no barramento AXI para prover acesso ao MicroBlaze. Para o algoritmo filtro de redução de ruído, foi avaliada somente a implementação com DMA, visto que já foi observado nos experimentos de processamento de sinal que a implementação com MMIO não oferece ganhos de desempenho. Esta implementação realiza acessos de 16 amostras de dados a cada requisição de leitura à memória.

A versão em software utilizada na ML605 e na BeagleBone implementa o mesmo algoritmo, utilizando aritmética em ponto flutuante de precisão simples (32 bits). Na BeagleBone, a versão em software é compilada fazendo uso das instruções SIMD disponíveis no processador.

Figura 5.4 – Exemplo de aplicação do algoritmo de redução de ruído

(a) Imagem original

(b) Imagem filtrada

Fonte: Autor

5.3.2 *rgb2yuv*

yuv é um espaço de cores diferente do tradicional *rgb*, que é tipicamente utilizado por sistemas de processamento de imagem. Uma das vantagens do uso do *yuv* é a possibilidade de compressão de componentes da imagem resultando em menos ruído para a percepção humana. Por isso, o *yuv* é necessário ao interfacear em dispositivos como televisores ou equipamentos fotográficos. Assim, existe a necessidade de conversão entre os espaços de cores *yuv* e *rgb* para interface entre diferentes aplicações, e se torna

interessante a sua aceleração ou implementação em hardware.

Na Equação (5.4), são apresentadas as operações realizadas sobre cada pixel para realizar a conversão de espaço de cores.

$$\begin{aligned} y &= 0.299r + 0.587g + 0.114b \\ u &= -0.147r - 0.288g + 0.436b + 128 \\ v &= 0.615r - 0.514g - 0.100b + 128 \end{aligned} \tag{5.4}$$

A Figura 5.5 mostra uma representação do sinal utilizado para testes com o algoritmo *rgb2yuv*.

Figura 5.5 – Exemplo de aplicação do algoritmo *rgb2yuv*

(a) Imagem original (em espaço de cores *rgb*)

(b) Imagem convertida (em representação do espaço de cores *yuv*)

Fonte: Autor

O algoritmo base para este estudo de caso em específico é disponível em (MATHWORKS, 2015c). Ambas as implementações em hardware e software para este estudo de caso foram geradas automaticamente através da ferramenta MATLAB®.

Na plataforma ML605, o bloco de processamento da conversão de cores *rgb2yuv* em hardware utiliza aritmética de ponto fixo, com números de 8 bits para cada compo-

nente de cor. Esta divisão de bits foi escolhida após a análise da ferramenta sobre o conjunto de dados a ser utilizado no experimento. O bloco de controle da conversão de cores *rgb2yuv* é gerado utilizando a ferramenta Xilinx EDK.

O periférico de hardware para a conversão de cores *rgb2yuv* é inserido no barramento AXI para prover acesso ao MicroBlaze. A implementação com acesso DMA realiza acessos de 16 amostras de dados a cada requisição de leitura à memória.

A versão em software utilizada na ML605 e na BeagleBone implementa o mesmo algoritmo, utilizando aritmética em ponto flutuante de precisão simples (32 bits). Na BeagleBone, a versão em software é compilada fazendo uso das instruções SIMD disponíveis no processador.

5.3.3 Resultados

A Tabela 5.3 mostra os resultados das variações testadas. Para o algoritmo Redução de Ruído, é utilizada uma imagem de alta resolução, tamanho 1920x1080. Para o algoritmo *rgb2yuv*, é utilizada uma imagem do tamanho 752x480. Para cada experimento, o algoritmo foi executado 100 vezes em cada variação, e o valor apresentado é a média dos tempos medidos.

Tabela 5.3 – Tempo de execução algoritmos Redução de ruído e *rgb2yuv*

	Tempo de execução (ms)		
		ML605	
Algoritmo	BeagleBone	Software	Hardware DMA
Redução de Ruído	4860	93530	140
rgb2yuv	1030	23510	200

Fonte: Autor

Nas implementações em software, cada amostra é processada em um software de aplicação no Linux. Na implementação em hardware DMA, toda a imagem é carregada no buffer DMA e repassada para processamento em hardware. O hardware gera uma interrupção quando todas as amostras são processadas e o buffer utilizado no DMA já contém a imagem final.

Neste experimento, é possível se observar a diferença em tempos de execução com o uso de DMA e grandes quantidades de dados. Conforme esperado, a vantagem do uso

do periférico DMA em hardware se torna mais aparente.

5.4 Aplicação em agricultura de precisão utilizando VANTs com processamento de imagens

O estudo de caso apresentado nesta seção é a aplicação desta arquitetura em um projeto de agricultura de precisão utilizando VANTs e processamento de imagens. Este estudo de caso é detalhado também em (ASSIS et al., 2016) e foi considerado durante o trabalho como uma das possíveis aplicações para esta plataforma. A aplicação deste estudo de caso para a plataforma descrita neste trabalho foi apresentada em (NUNES; BEHNCK; PEREIRA, 2015).

Três plataformas foram utilizadas: a plataforma Desktop, a plataforma Raspberry Pi, e a plataforma ML605.

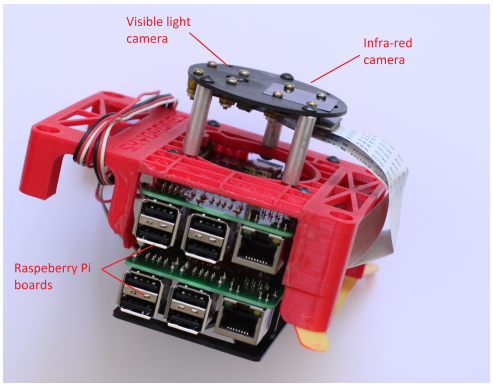
Os experimentos consistem da análise do tempo de execução do periférico de processamento de imagens, seguido por uma análise de implementação com Agentes e migração de Agentes em hardware.

5.4.1 Descrição

VANTs são utilizados como plataformas de sensores que atingem baixas altitudes, para a obtenção de imagens de plantações. Nas imagens obtidas, é possível se obter características da região observada através de algoritmos de processamento de imagem. A principal vantagem deste tipo de sistemas é o baixo custo e facilidade de implementação, comparado a outras soluções como sensores baseados em satélites. Em adição às unidades presentes nos dispositivos VANTs comerciais, que geralmente são responsáveis por controle de voo e navegação, pode ser necessário adicionar novas unidades de processamento. Estas unidades de processamento adicionais podem ser projetadas para acrescentar funções avançadas como de processamento de imagem e planejamento de trajetória para o VANT. Um exemplo de um módulo de controle que pode ser acrescentado a um VANT é apresentado na Figura 5.6.

Mesmo os sensores comerciais padrão atuais são capazes de prover imagens de alta resolução e em quantidades arbitrárias, o que permite que se obtenha grandes quantidades de dados numa única missão de voo (ZHANG; KOVACS, 2012). A limitação res-

Figura 5.6 – Exemplo de módulo de controle composto por dois computadores Raspberry Pi com câmeras e sensores infravermelho



Fonte: (ASSIS et al., 2016)

tante, então, é relacionada à capacidade de analisar estas quantidades de dados de forma eficiente para que sejam obtidas informações úteis a partir destes dados. No entanto, os algoritmos para processamento dessas imagens podem demandar alto poder de processamento, e executá-los em módulos embarcados nos VANTs pode impactar o seu consumo de energia. O tempo de execução destes algoritmos também é muito importante, principalmente se este tempo de processamento é importante para a aplicação. Um exemplo disso é o caso de o resultado do processamento influenciar o traçado de trajetória de voo dos VANTs. Se a redução do tempo de processamento dos VANTs for um dos principais objetivos, é importante notar que este tipo de algoritmo pode ser tipicamente acelerado em hardware. Neste caso, é interessante analisar se o uso de um FPGA é vantajoso para a aplicação.

Cenários com múltiplos VANTs podem beneficiar de implementações com SMAs. Por exemplo, alguns autores abordam o problema de coordenação de grupos de VANTs durante missões de busca ou sensoriamento utilizando Agentes (SCHLECHT et al., 2003) (KINGSTON; BEARD; HOLT, 2008).

Neste estudo de caso, um Agente com processamento de imagem é estudado, que pode estar a bordo de um VANT e prover informações para outros Agentes com papéis diferentes. Por exemplo, para um Agente responsável por traçar caminhos ou outro Agente com sensores. Algumas possibilidades estudadas são a de migração deste Agente para outra plataforma e a possibilidade de utilizar reconfiguração parcial de um FPGA, quando a plataforma suporta este dispositivo. A migração de Agentes oferece capacidades interessantes ao sistema pois permite que um Agente seja substituído no caso de eventos de hardware (como bateria fraca ou detecção de falhas de hardware) para que ele possa

continuar executando suas tarefas sem perder o estado. O uso de reconfiguração parcial permite ainda que um dado Agente possa hospedar mais de um Agente executando em hardware em um único FPGA.

O algoritmo de processamento de imagem utilizado para avaliar o framework executa o processo de segmentação em imagens de uma plantação de trigo jovem, diferenciando porções que representam solo e plantação. A primeira etapa da segmentação é a conversão de um espaço de cores RGB para um espaço de cores com *excesso de verde* (WOEBBECKE et al., 1995), baseado na Equação (5.5),

$$ex_g = 2g - r - b \tag{5.5}$$

onde r, g e b são respectivamente os canais vermelho, verde e azul da imagem. Após o cálculo do índice, um filtro 5×5 com uma matriz com pesos homogêneos é aplicado sobre a imagem resultante. Por fim, os pixels representando plantação e solo são segmentados através de aplicação de um limiar simples na intensidade do pixel.

Este algoritmo simples pode ser utilizado para detectar zonas do cultivo que tiveram pouco desenvolvimento. A razão entre o total de "pixels verdes" e "pixels de solo" é um indicativo da cobertura de plantação no *frame*, onde valores baixos podem indicar problemas na região analisada. A Figura 5.7 apresenta a representação da imagem antes (Figura 5.8(a)) e após (Figura 5.8(b)) o algoritmo de processamento.

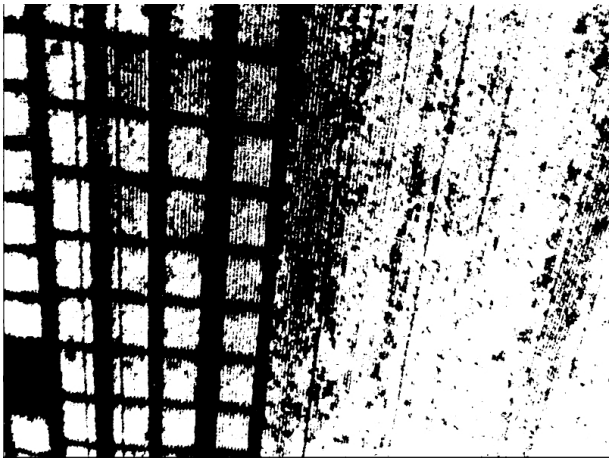
Na plataforma ML605, o bloco de processamento de imagem em hardware utiliza aritmética de ponto fixo, com números de 8 bits para cada componente de cor. Esta divisão de bits foi escolhida após a análise da ferramenta sobre o conjunto de dados a ser utilizado no experimento. O bloco de controle da processamento de imagem é gerado utilizando a ferramenta Xilinx EDK.

O periférico de hardware para o processamento de imagem é inserido no barramento AXI para prover acesso ao MicroBlaze. A implementação com acesso DMA realiza acessos de 16 amostras de dados a cada requisição de leitura à memória.

A versão em software utilizadas em todas as plataformas implementa o mesmo algoritmo, utilizando aritmética em ponto flutuante de precisão simples (32 bits). Para a Raspberry Pi e para o Desktop, as versões em software são compiladas fazendo uso das instruções SIMD disponíveis no processador.

Figura 5.7 – Exemplo de aplicação do algoritmo de segmentação

(a) Exemplo de imagem de plantação. À esquerda, trechos retangulares de plantação de trigo. À direita, uma região contínua com regiões degradadas



(b) Representação do resultado do algoritmo de segmentação.

Fonte: (ASSIS et al., 2016)

5.4.2 Resultados

A Tabela 5.4 apresenta os resultados de tempo de execução do algoritmo de processamento de imagem nos diferentes sistemas. O tempo medido corresponde ao tempo para execução do processamento da imagem, excluindo os tempos de abertura de arquivo ou tempo de captura da imagem. Além das diferentes implementações (hardware ou software), os resultados foram medidos com três implementações diferentes que trabalham com imagens de diferentes resoluções. As resoluções testadas são 256x256, 1920x1080 e 2592x1944. A maior resolução escolhida, 2592x1944, foi utilizada pois é a resolução da

câmera 5MP da Raspberry Pi.

Tabela 5.4 – Tempo de execução para diferentes resoluções do algoritmo de processamento de imagem

	Tempo de execução (ms)		
Resolução	Raspberry Pi	Desktop	ML605 Hardware DMA
256x256	110	4	7.99
1920x1080	4290	131	21.87
2592x1944	10510	333	42.01

Fonte: Autor (NUNES; BEHNCK; PEREIRA, 2015)

As implementações em software foram executadas como aplicações Linux, e portanto as medidas apresentam a média de várias execuções, para reduzir o ruído devido à concorrência com outros processos. A implementação no FPGA apresenta o melhor tempo de execução. Algumas outras observações podem ser feitas quanto aos resultados.

Os resultados com a Raspberry Pi mostram que a plataforma é capaz de realizar processamento de imagem de forma relativamente rápida para imagens de baixa resolução, e poderia ser utilizada em aplicações que não requerem processamento de imagem em alta resolução e em baixa taxa de imagens a processar.

É notável que os tempos de processamento no FPGA sejam consideravelmente mais baixos do que o processamento em software no Desktop para imagens de mais alta resolução, mesmo considerando que o FPGA está executando a somente 100MHz. Deve ser considerado que, para este experimento, o periférico AXI utilizado na implementação em hardware é capaz de realizar acessos DMA em rajadas, e realiza o processamento em paralelo com os acessos à memória. O periférico é capaz de realizar uma iteração do filtro com multiplicação de matrizes a cada ciclo, após ter o seu pipeline cheio.

O framework JADE também provê a funcionalidade de migração de Agentes, que pode ser explorada na plataforma ML605 em conjunto com o uso de reconfiguração parcial do FPGA.

Ainda neste experimento, o algoritmo foi implementado também de forma que a aplicação é capaz de alterar o hardware numa área reconfigurável. A aplicação é capaz de escolher entre as implementações que atendem as três resoluções diferentes do algoritmo de processamento de imagem. Este experimento também explora medidas do tempo de reconfiguração parcial para executar a troca de algoritmo.

Conforme mencionado na Seção 4.1, o tamanho e tempo de reconfiguração das regiões de reconfiguração parcial variam de acordo com o tamanho da região de reconfiguração parcial reservada.

Para uma partição com tamanho suficiente para atender a implementação do algoritmo para a resolução de 2592x1944, o tempo de reconfiguração parcial foi medido como aproximadamente **500ms**.

A região reconfigurável foi reservada com aproximadamente três vezes o tamanho mínimo necessário, para facilitar a geração do *bitstream* pelas ferramentas de projeto. Este tempo foi medido na plataforma ML605 realizando reconfiguração parcial no bloco descrito na Tabela 5.5. Na Tabela 5.5 são apresentados os recursos necessários e área da partição para diferentes resoluções do algoritmo de processamento de imagem no FPGA. Os recursos apresentados são recursos reconfiguráveis do FPGA como LUTs (Lookup Tables), Flip-flops, multiplexadores (Xilinx, Inc., 2013). A partição reservada possui os recursos na coluna Disponível, e as outras colunas mostram o uso de recursos da implementação em cada resolução.

Tabela 5.5 – Recursos necessários e área da partição para diferentes resoluções do algoritmo de processamento de imagem no FPGA

		Resolução					
		256x256		1920x1080		2592x1944	
Recurso	Disponível	Utilizado	% util.	Utilizado	% util.	Utilizado	% util.
LUT	12480	1651	14	4367	35	4481	36
FD_LD	24960	711	3	1372	6	1378	6
SLICEL	1680	221	14	586	35	602	36
SLICEM	1440	193	14	506	36	520	37
RAMBFIFO36E1	36	2	6	5	14	10	28

Fonte: Autor

O trabalho semelhante foi apresentado em (CEMIN; GOTZ; PEREIRA, 2012) que apresenta tempos da ordem de **20 segundos** para migração de Agentes, incluindo reconfiguração dinâmica do FPGA. O uso de reconfiguração parcial é, portanto, um ganho considerável, e a redução de uma ordem de grandeza no tempo de reconfiguração pode possibilitar novas aplicações neste cenário. Dado que uma área relativamente grande do FPGA foi reservada, e considerando que outras aplicações podem ser atendidas com áreas menores reservadas para reconfiguração parcial, é possível se considerar aplicações de Agentes que realizam reconfiguração durante a execução. O algoritmo de processamento

de imagem apresentada é um exemplo de aplicação, onde pode-se reconfigurar o hardware para operar de forma ótima em um determinado tipo de imagem.

5.5 Migração de Agentes JADE com reconfiguração parcial

Este experimento faz uso mais extensivo das funcionalidades do JADE, explorando melhor vantagens de utilização de Agentes na plataforma proposta.

5.5.1 Descrição

O experimento consiste da utilização de duas plataformas embarcadas executando o JADE, de forma a permitir a migração de Agentes entre as plataformas embarcadas. O experimento utiliza também uma terceira plataforma na rede, que é a plataforma de gerência, e que é responsável por monitorar as ações dos Agentes na rede.

5.5.2 Resultados

Este experimento utiliza a plataforma ML605 e a plataforma BeagleBone.

O experimento consiste de medidas de tempo do Agente HwReconfRequesterAgent que é submetido à migração entre as duas plataformas. Conforme introduzido na Seção 4.4, este experimento leva em conta a existência de um Agente que gerencia as regiões reconfiguráveis de hardware em cada plataforma, denominado HwReconfAgent. Portanto, a cada migração do Agente, o Agente migrado troca mensagens JADE com o Agente HwReconfRequesterAgent da nova plataforma para saber se pode ou não utilizar uma região reconfigurável.

A Tabela 5.6 apresenta os resultados de tempo de migração do Agente HwReconfRequesterAgent neste cenário com reconfiguração parcial.

O tempo Migração (JADE) representa o tempo ocupado pelo JADE para realizar a migração do Agente e começar a execução do mesmo na plataforma alvo. O tempo de Reprogramação representa o tempo de comunicação do Agente migrado com o HwReconfAgent, e inclui o tempo de reprogramação no hardware, caso disponível. Para o experimento, assume-se que ao migrar para a plataforma ML605, a região reconfigurável está disponível, portanto o tempo Migração + solicitação de reconfiguração representa

Tabela 5.6 – Tempo de migração de Agentes JADE com reconfiguração parcial

	Tempo de migração (ms)	
Plataforma alvo	Migração (JADE)	Migração + solicitação de reconfiguração
ML605 → BeagleBone	368.5	417.8
BeagleBone → ML605	889.2	1035.5

Fonte: Autor

sempre a comunicação e a reprogramação no hardware. Na migração para a plataforma BeagleBone, não existe a possibilidade de reconfiguração de hardware, então o tempo adicional é somente a comunicação com o HwReconfAgent local.

Com este experimento, é possível verificar que, dado o baixo custo de reconfiguração do hardware com o uso de reconfiguração parcial, o custo adicional da reconfiguração não chega a ser um *overhead* extremamente significativo quando comparado ao tempo de migração do próprio JADE entre plataformas embarcadas.

5.6 Outros experimentos

Esta seção apresenta outros experimentos realizados com o uso do MicroBlaze e de reconfiguração parcial, que complementam os estudos de caso.

5.6.1 Resultados do uso de reconfiguração parcial

Medidas do tempo de reconfiguração parcial foram feitas através de requisições isoladas ao driver HWICAP na plataforma ML605, e são apresentados na Tabela 5.7. O tempo reportado na tabela é a média de 100 requisições para cada partição. Os tempos foram medidos através de instrumentação do driver HWICAP. Este experimento foi apresentado inicialmente em (NUNES; PEREIRA, 2015).

Detalhes sobre o uso de recursos em cada partição podem ser vistos na Tabela 5.8. Uma representação visual das zonas de reconfiguração parcial foi apresentada na Figura 2.5.

Conforme o esperado, os tempos de reconfiguração parcial e tamanho dos bitstreams aumenta de acordo com a quantidade de recursos utilizados no FPGA. Dado que o tempo de reconfiguração é da ordem de milissegundos para reconfigurar uma partição do

Tabela 5.7 – Tempo de reconfiguração parcial com HWICAP

Partição	Tamanho (bytes)	Tempo de reconfiguração (ms)
math	68413	53.6
fir	678565	289.2

Fonte: Autor (NUNES; PEREIRA, 2015)

Tabela 5.8 – Tamanho e utilização de recursos nas partições reconfiguráveis

	Implementação		Tamanho da região reconfigurável	
Partição	Flip-flops	LUTs	Flip-flops	LUTs
math	32	32	400	800
fir	1410	1420	5472	10944

Fonte: Autor

FPGA mesmo com uma partição ocupando uma área relativamente grande no FPGA, é viável explorar esta funcionalidade para aplicações de Agentes em hardware. Para algumas aplicações, é possível considerar a reconfiguração parcial como um recurso que pode ser explorado com relativa frequência.

5.6.2 Responsividade do soft core MicroBlaze em comunicação com o JADE

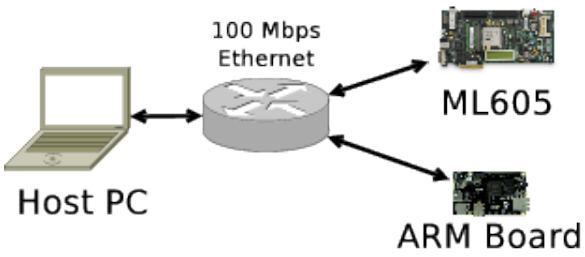
Nestes experimentos, uma rede composta de três plataformas executando o JADE foi construída. A rede de testes é representada na Figura 5.8. As plataformas utilizadas neste teste são a plataforma ML605, o Desktop e a Cubieboard2. Este experimento foi apresentado inicialmente em (NUNES; PEREIRA, 2015).

Os testes realizados neste cenário foram de responsividade, tempo de reconfiguração de hardware, envolvendo comunicação entre Agentes JADE.

O objetivo deste teste é avaliar o impacto no desempenho do sistema ao se utilizar o processador MicroBlaze na plataforma ML605 com o JADE.

Os testes de responsividade foram realizados através da execução de Agentes JADE customizados para testes. Cada Agente de testes consiste de um Agente que aguarda uma mensagem com conteúdo específico, e responde com uma mensagem semelhante imediatamente ao recebê-la. Na plataforma ML605, variações do Agente de testes realizam também acessos a um periférico em hardware via MMIO ou reconfigu-

Figura 5.8 – Rede para realização do experimento de responsividade



Fonte: Autor (NUNES; PEREIRA, 2015)

ração de hardware através do HWICAP. Com o Agente de testes executando em cada uma das plataformas embarcadas, um outro Agente JADE é instanciado no Desktop para iniciar uma comunicação com os Agente de testes. Os tempos de envio da mensagem de teste e resposta são medidos imediatamente após o envio e resposta. Os testes foram realizados separadamente com cada Agente de testes, e os tempos apresentados são obtidos com a média dos tempos entre 1000 mensagens trocadas.

A Tabela 5.9 mostra a média de tempos de respostas observados. Os resultados mostram os tempos de resposta do MicroBlaze para simples resposta, resposta após acessar um periférico de hardware (acesso HW), e resposta após realizar uma operação de reconfiguração parcial. O bloco de reconfiguração parcial utilizado é o mesmo bloco math apresentado na Tabela 5.8.

Tabela 5.9 – Tempo de resposta no experimento de responsividade

Plataforma	Tempo de resposta JADE (ms)
Cubieboard2	6
ML605	81
ML605 + Acesso HW	104
ML605 + Reconfiguração	154

Fonte: Autor

A Tabela 5.9 mostra que o poder de processamento reduzido do soft core pode ter impacto significativo na responsividade para mensagens JADE, quando o tempo de processamento da aplicação é baixo. No entanto, nos casos onde o tempo de processamento absoluto da aplicação é maior, este tempo pode se tornar menos relevante. O esperado é que a aplicação consiga ter um ganho no tempo de execução total, o que pode se tornar mais factível para casos como os apresentados na Seção 5.3.3.

6 CONCLUSÃO

O uso de Agentes mostra ser um tema ainda no campo de pesquisa, e fica evidente pelos trabalhos estudados que alguns pontos ainda podem ser melhor evoluídos. É considerado que atualmente as aplicações alvo para a tecnologia de Agentes são de logística e transporte, controle de manufatura, fábricas de montagem e aplicações militares e de defesa. Outras aplicações observadas com frequência em estudos são de sistemas de distribuição de energia e construções inteligentes. Tópicos como uso de Agentes em plataformas móveis com sistemas operacionais móveis como Android e IOT já são citados em publicações relacionadas a SMAs.

Conforme apontado pelos trabalhos no Capítulo 3, restam alguns desafios como o controle de tempo-real em SMAs e mudanças na forma de pensamento da indústria em geral para adoção desta tecnologia. Alguns pontos que eram observados anteriormente como desafios, como a padronização de frameworks para o desenvolvimento, parecem estar convergindo para uma solução. É possível observar que boa parte dos trabalhos já opta por frameworks padronizados como o JADE, que oferecem facilidades de implementação além da padronização.

A principal contribuição deste trabalho é a proposta da arquitetura e implementação da plataforma para SMAs envolvendo hardware reconfigurável, no Capítulo 4.

Neste trabalho, foi proposta uma plataforma para SMAs envolvendo hardware reconfigurável. A arquitetura da plataforma foi apresentada na Seção 4.1. A plataforma foi detalhada de um ponto de vista prático e foi validada através de estudos de caso, apresentados no Capítulo 5. Os estudos de caso incluem implementações de Agentes em software e em hardware. Os algoritmos e implementações da plataforma e dos estudos de caso foram detalhados e os experimentos foram realizados em diversas plataformas para comparação, incluindo a plataforma com hardware reconfigurável. Foi avaliado também o uso da plataforma proposta em uma aplicação de processamento de imagem embarcado utilizando VANTs.

A plataforma proposta utiliza o framework JADE, que é um dos frameworks mais populares no estado da arte de desenvolvimento de SMA. A plataforma proposta é capaz de executar o JADE mesmo com Agentes implementados em hardware através do uso do processador soft core MicroBlaze. O processador soft core se mostrou capaz de executar o framework JADE com o uso de uma JVM simplificada, provendo recursos suficientes para executar os Agentes e interfacear com o FPGA de forma satisfatória. O uso deste

processador soft core pode agregar valor à solução final pois elimina a necessidade de inclusão de um processador adicional para fazer a interface entre o FPGA e o framework de Agentes. Além disso, o uso do soft core dentro do FPGA situado próximo aos periféricos de processamento facilita o uso de técnicas de otimização de transferência de dados, como o uso de DMA.

O uso do JADE apresentou impacto perceptível de desempenho nas plataformas com menos recursos, e sua aplicação pode ser reavaliada em implementações futuras da plataforma proposta. O MicroBlaze oferece um ambiente completo para a execução de Agentes mesmo com recursos reduzidos de hardware, porém os experimentos no Capítulo 5 mostram que a responsividade é consideravelmente impactada, quando comparado a um processador em um sistema embarcado utilizando um SOC moderno. Uma alternativa para eliminar esta desvantagem, caso se torne um fator limitante para a aplicação, é avaliar o uso de um outro framework de Agentes mais leve. Apesar disso, o uso JADE traz vantagens significativas para a implementação, oferecendo funcionalidades como a migração de Agentes, que é vantajosa em conjunto com o uso de reconfiguração parcial. Através do uso da reconfiguração parcial em conjunto com o JADE, é possível se implementar mais de um Agente em hardware ao mesmo tempo e de forma reconfigurável, o que também é uma vantagem com relação aos trabalhos anteriores.

Uma outra contribuição deste trabalho consiste do estudo do uso da reconfiguração parcial de hardware em plataformas de Agentes heterogêneas, com uma implementação real. O uso de reconfiguração parcial de FPGA foi implementado e avaliado nos estudos de caso. Os tempos de reconfiguração parcial são significativamente menores do que os de reconfiguração completa do FPGA. Além disso, o uso de regiões reconfiguráveis permite a implementação de múltiplos Agentes em hardware em um único FPGA. Estes benefícios já haviam sido previstos em trabalhos anteriores, porém neste trabalho foi possível validar que o uso de reconfiguração parcial em conjunto com uma aplicação de alto nível como Agentes é viável com ferramentas e tecnologias existentes.

O uso de operações com DMA se mostrou determinante para maximizar o ganho de desempenho em aplicações como processamento de imagem, pois reduz o *overhead* da transferência de dados que, de outra forma, poderia anular os ganhos obtidos com o processamento em FPGA. O uso desta técnica é um diferencial neste trabalho em relação aos outros trabalhos que propõe o uso de Agentes com processamento em hardware, pois alavanca o potencial de processamento do FPGA para computação com grandes quantidades de dados, como imagens de alta resolução.

Os experimentos realizados apresentam um ganho significativo de desempenho nas implementações em FPGA, mesmo considerando que o FPGA executa em frequências reduzidas em comparação aos processadores testados. Em aplicações com processamento de imagem, este ganho pode oferecer vantagens nas aplicações, como possibilitar um maior número de imagens processadas por unidade de tempo. Deve-se levar em conta que os processadores testados também oferecem capacidades de aceleração deste tipo de computação, através do uso de instruções SIMD, por exemplo. Em alguns casos, foi observado que a execução em software com processadores modernos é mais rápida do que com hardware dedicado. Isto ocorre dado o *overhead* de comunicação e transferência de dados entre o processador e o FPGA. Assim, é observado que o uso de Agentes em hardware só resulta em reais benefícios caso a quantidade de dados a serem processados seja significativa.

É importante ressaltar que os resultados apresentados no Capítulo 5 foram obtidos através de medidas com implementações reais. Alguns dos estudos de caso requerem uma implementação não trivial, como o uso de técnicas avançadas para implementação de periféricos, e uso de reconfiguração parcial. Além disso, os experimentos foram realizados com a implementação de drivers Linux e o *overhead* do SO. Assim, os resultados de tempo de execução com o detalhamento da implementação se tornam uma contribuição adicional deste trabalho, pois podem servir de referência para decisões futuras sobre implementações semelhantes.

Neste trabalho, não foi feita uma análise de consumo de energia das implementações realizadas, que é relevante considerando o contexto de sistemas embarcados. As medidas de desempenho apontam para que a plataforma ofereça também uma redução no consumo de energia, visto que executam em considerável menos tempo, porém isso deve ser avaliado com implementação no hardware e é deixado como trabalho futuro.

Nos estudos de caso apresentados, foi avaliado o uso do soft core MicroBlaze. Foi observado que o uso deste processador teve impacto na responsividade dos Agentes. Uma alternativa ao uso do MicroBlaze seria o uso de processadores SOC FPGA, que processador embarcado no mesmo chip. Os SOC FPGA da linha Zynq, da Xilinx, incluem processadores ARM de até dois núcleos operando até 1GHz. Mesmo considerando que o uso do soft core foi um dado importante do trabalho, a comparação com um SOC FPGA seria interessante em um trabalho futuro. Um exemplo de plataforma de desenvolvimento é a Xilinx MicroZed, que inclui o FPGA Zynq-7010, com um processaor ARM Cortex-A9 de dois núcleos operando a 667MHz e 1GB de memória DDR3. Para isso,

pode-se também considerar que os periféricos AXI são também compatíveis com estas plataformas.

Outro estudo de caso que pode ser realizado como trabalho futuro é a comparação destes resultados com resultados de execução dos algoritmos em unidades de processamento gráfico. Mesmo em sistemas embarcados, é crescente a disponibilidade de SOCs com unidades de processamento gráfico capazes de realizar processamento de propósito geral.

Outras tecnologias de software do Linux também podem ser avaliadas para avanços futuros neste trabalho. Versões recentes de Linux oferecem frameworks para interação com FPGAs, cujo uso pode ser analisado para melhorias nesta interface. Versões recentes de Linux oferecem também uma API para compartilhamento de buffers de DMA chamada dma-buf, que pode ser analisada como forma de reduzir o esforço na implementação dos drivers Linux.

REFERÊNCIAS

ALTERA CORPORATION. **Architecture Matters: Choosing the Right SoC FPGA for Your Application**. 2013.

ASPENCORE. **2017 Embedded Markets Study**. 2017.

ASSIS, L. F. et al. Dynamic sensor management: Extending sensor web for near real-time mobile sensor integration in dynamic scenarios. 03 2016.

BELKACEMI, R. et al. Multi-agent systems hardware development and deployment for smart grid control applications. In: **Power and Energy Society General Meeting, 2011 IEEE**. [S.l.: s.n.], 2011. p. 1–8. ISSN 1944-9925.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. Jade: A fipa2000 compliant agent development environment. In: **Proceedings of the Fifth International Conference on Autonomous Agents**. New York, NY, USA: ACM, 2001. (AGENTS '01), p. 216–217. ISBN 1-58113-326-X.

BOSSE, S. Distributed agent-based computing in material-embedded sensor network systems with the agent-on-chip architecture. **Sensors Journal, IEEE**, v. 14, n. 7, p. 2159–2170, July 2014. ISSN 1530-437X.

BUILDROOT. **Buildroot: Making Embedded Linux Easy**. 2018. <https://buildroot.org/>. Acesso em: 26 de Junho de 2018.

CAIRE., G. **JADE Tutorial: JADE Programming for Beginners**. 2009.

CEMIN, D.; GOTZ, M.; PEREIRA, C. Reconfigurable agents for heterogeneous wireless sensor networks. In: **Computing System Engineering (SBESC), 2012 Brazilian Symposium on**. [S.l.: s.n.], 2012. p. 1–6. ISSN 2324-7886.

CEMIN, D.; PEREIRA, C. E. **Arquitetura de agentes móveis reconfiguráveis para redes de sensores sem fio**. 2012. Dissertação (Mestrado). PPGEE, Universidade Federal do Rio Grande do Sul.

COMMONS, W. **Simplified Structure of the Linux Kernel**. 2015. <https://commons.wikimedia.org/wiki/File:Simplified_Structure_of_the_Linux_Kernel.svg>. Acesso em: 26 de Junho de 2018.

DAVIDSSON, P.; BOMAN, M. A multi-agent system for controlling intelligent buildings. In: **MultiAgent Systems, 2000. Proceedings. Fourth International Conference on**. [S.l.: s.n.], 2000. p. 377–378.

ENEA. **Enea Linux for high throughput and low latency in NFV applications | Enea**. 2018. <https://www.enea.com/products/operating-systems/enea-linux/>. Acesso em: 26 de Junho de 2018.

ESTRADA, D.; LEE, K. Multi-agent system implementation in jade environment for power plant control. In: **Power and Energy Society General Meeting (PES), 2013 IEEE**. [S.l.: s.n.], 2013. p. 1–5. ISSN 1944-9925.

FILGUEIRAS, T.; LUNG, L. C.; RECH, L. de O. Providing real-time scheduling for mobile agents in the jade platform. In: **Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2012 IEEE 15th International Symposium on**. [S.l.: s.n.], 2012. p. 8–15. ISSN 1555-0885.

Free Software Foundation, Inc. **GNU Classpath - GNU Project - Free Software Foundation (FSF)**. 2018. <https://www.gnu.org/software/classpath/home.html/>. Acesso em: 26 de Junho de 2018.

GHOSN, S. et al. Agent-oriented designs for a self healing smart grid. In: **Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on**. [S.l.: s.n.], 2010. p. 461–466.

GODDEMEIER, N.; BEHNKE, D.; WIETFELD, C. Impact of communication constraints on consensus finding in multi-agent systems. In: **Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on**. [S.l.: s.n.], 2013. p. 2464–2468. ISSN 2166-9570.

Intel Corporation. **Avalon Interface Specifications**. 2017.

Intel Corporation. **Intel Arria 10 Device Overview**. 2018.

JENNINGS, N. R.; SYCARA, K.; WOOLDRIDGE, M. A roadmap of agent research and development. **Autonomous Agents and Multi-Agent Systems**, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, n. 1, p. 7–38, jan. 1998. ISSN 1387-2532.

JUNEJA, D.; JAGGA, A.; SINGH, A. A review of FIPA standardized agent communication language and interaction protocols. **Journal of Network Communications and Emerging Technologies**, v. 5, n. 2, p. 179–191, Dec 2015. ISSN 2395-5317.

KAINDL, H.; VALLEE, M.; ARNAUTOVIC, E. Self-representation for self-configuration and monitoring in agent-based flexible automation systems. **Systems, Man, and Cybernetics: Systems, IEEE Transactions on**, v. 43, n. 1, p. 164–175, Jan 2013. ISSN 2168-2216.

KALMAN, R. E. A new approach to linear filtering and prediction problems. **Transactions of the ASME–Journal of Basic Engineering**, v. 82, n. Series D, p. 35–45, 1960.

KINGSTON, D.; BEARD, R. W.; HOLT, R. S. Decentralized perimeter surveillance using a team of uavs. **IEEE Transactions on Robotics**, v. 24, n. 6, p. 1394–1404, 2008.

KIRAN, P.; CHANDRAKALA, K. R. M. V.; NAMBIAR, T. N. P. Multi-agent based systems on micro grid – a review. In: **2017 International Conference on Intelligent Computing and Control (I2C2)**. [S.l.: s.n.], 2017. p. 1–6.

KROL, D.; NOWAKOWSKI, F. Practical performance aspects of using real-time multi-agent platform in complex systems. In: **Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on**. [S.l.: s.n.], 2013. p. 1121–1126.

KRUGER, C. P.; HANCKE, G. P. Benchmarking internet of things devices. In: **2014 12th IEEE International Conference on Industrial Informatics (INDIN)**. [S.l.: s.n.], 2014. p. 611–616. ISSN 1935-4576.

KULASEKERA, A. L. et al. A review on multi-agent systems in microgrid applications. In: **Innovative Smart Grid Technologies - India (ISGT India), 2011 IEEE PES**. [S.l.: s.n.], 2011. p. 173–177.

LEITAO, P.; MARIK, V.; VRBA, P. Past, present, and future of industrial agent applications. **Industrial Informatics, IEEE Transactions on**, v. 9, n. 4, p. 2360–2372, Nov 2013. ISSN 1551-3203.

LIU, C. Implementation of a highly scalable blokus duo solver on fpga. In: **Field-Programmable Technology (FPT), 2013 International Conference on**. [S.l.: s.n.], 2013. p. 482–485.

LOUGHER, R. **JamVM – A compact Java Virtual Machine**. 2014. <http://jamvm.sourceforge.net/>. Acesso em: 26 de Junho de 2018.

LYONS, R. G. **Understanding Digital Signal Processing, 2nd Edition**. 2. ed. [S.l.]: Prentice Hall, 2004. ISBN 0131089897,9780131089891.

MARIK, V.; LAZANSKY, J. Industrial applications of agent technologies. **Control Engineering Practice**, v. 15, n. 11, p. 1364 – 1380, 2007. ISSN 0967-0661. Special Issue on Manufacturing Plant Control: Challenges and Issues INCOM 2004 11th IFAC INCOM04 Symposium on Information Control Problems in Manufacturing.

MASSAWE, L.; KINYUA, J.; AGHDASI, F. An implementation of a multi-agent based rfid middleware for asset management system using the jade platform. In: **IST-Africa, 2010**. [S.l.: s.n.], 2010. p. 1–8.

MATHWORKS. **Fixed-Point Type Conversion and Refinement**. 2015. <http://www.mathworks.com/help/hdlcoder/examples/fixed-point-type-conversion-and-refinement.html>. Acesso em: 26 de Junho de 2018.

MATHWORKS. **Getting Started with MATLAB to HDL Workflow**. 2015. <http://www.mathworks.com/help/hdlcoder/examples/getting-started-with-matlab-to-hdl-workflow.html>. Acesso em: 26 de Junho de 2018.

MATHWORKS. **Image Format Conversion: RGB to YUV**. 2015. <http://www.mathworks.com/help/hdlcoder/examples/image-format-conversion-rgb-to-yuv.html>. Acesso em: 26 de Junho de 2018.

MATHWORKS. **Noise Removal and Image Sharpening**. 2015. <http://www.mathworks.com/help/hdlcoder/examples/noise-removal-and-image-sharpening.html>. Acesso em: 26 de Junho de 2018.

MATTSSON, D.; CHRISTENSSON, M. **Evaluation of synthesizable CPU cores**. Dissertação (Mestrado) — Chalmer's University of Technology, 2004.

MEGHERBI, D.; KIM, M.; MADERA, M. A study of collaborative distributed multi-goal & multi-agent-based systems for large critical key infrastructures and resources (ckir) dynamic monitoring and surveillance. In: **Technologies for Homeland Security (HST), 2013 IEEE International Conference on**. [S.l.: s.n.], 2013. p. 687–692.

MENG, Y. An agent-based mobile robot system using configurable soc technique. In: **Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on**. [S.l.: s.n.], 2006. p. 3368–3373. ISSN 1050-4729.

MOIS, G. et al. Reconfiguration and hardware agents in testing and repair of distributed systems. In: **Design Test Symposium (EWDTS), 2010 East-West**. [S.l.: s.n.], 2010. p. 195–198.

MONTAVISTA. **Embedded Linux Solutions for Accelerating Secure IoT, NFV & 5G Deployment: Montavista**. 2018. <http://www.mvista.com/>. Acesso em: 26 de Junho de 2018.

MUSTAPHA, K.; MCHEICK, H.; MELLOULI, S. Modeling and simulation agent-based of natural disaster complex systems. **Procedia Computer Science**, v. 21, n. 0, p. 148 – 155, 2013. ISSN 1877-0509. The 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013).

NAJI, H. Implementing data flow operations with multi hardware agent systems. In: **System Theory, 2003. Proceedings of the 35th Southeastern Symposium on**. [S.l.: s.n.], 2003. p. 227–231. ISSN 0094-2898.

NAJI, H. Sensor fusion using hardware agents, an implementation example. In: **Industrial Informatics, 2005. INDIN '05. 2005 3rd IEEE International Conference on**. [S.l.: s.n.], 2005. p. 378–383.

NUNES Érico de M.; BEHNCK, L.; PEREIRA, C. E. Multi-agent based implementation of an embedded image processing system in fpga for precision agriculture using uavs. **IESS 2015 (IFIP International Embedded Systems Symposium)**, 2015.

NUNES Érico de M.; PEREIRA, C. E. Reconfgurable hardware agents using jade and fpga partial reconfguration. **APRES 2015 (7th Workshop on Adaptive and Reconfgurable Embedded Systems)**, 2015.

NWANA, H. S. et al. Zeus: A toolkit for building distributed multi-agent systems. **Applied Artifical Intelligence Journal**, v. 13, p. 129–186, 1999.

O'BRIEN, P. D.; NICOL, R. C. Fipa — towards a standard for software agents. **BT Technology Journal**, v. 16, n. 3, p. 51–59, Jul 1998. ISSN 1573-1995. Disponível em: <https://doi.org/10.1023/A:1009621729979>.

OPEN EMBEDDED. **Openembedded.org**. 2018. <https://www.openembedded.org/>. Acesso em: 26 de Junho de 2018.

PEIXOTO, J. A.; PEREIRA, C. E. **Sistema minimamente invasivo baseado em agentes aplicado em controladores lógicos programáveis**. 2016. Tese (Doutorado). PPGEE, Universidade Federal do Rio Grande do Sul.

PEREIRA, A.; RODRIGUES, N.; LEITAO, P. Deployment of multi-agent systems for industrial applications. In: **Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on**. [S.l.: s.n.], 2012. p. 1–8. ISSN 1946-0740.

PEREIRA, C. E.; CARRO, L. Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control. **Annual Reviews in Control**, v. 31, n. 1, p. 81 – 92, 2007. ISSN 1367-5788.

PERRAJU, T. Multi-agent architectures for high assurance systems. In: **American Control Conference, 1999. Proceedings of the 1999**. [S.l.: s.n.], 1999. v. 5, p. 3154–3157 vol.5. ISSN 0743-1619.

RASPIAN. **FrontPage - Raspian**. 2018. <https://www.raspbian.org/>. Acesso em: 26 de Junho de 2018.

RIVER, W. **Wind River Linux**. 2018. <https://www.windriver.com/products/linux/>. Acesso em: 26 de Junho de 2018.

ROSCIA, M.; LONGO, M.; LAZAROIU, G. Smart city by multi-agent systems. In: **Renewable Energy Research and Applications (ICRERA), 2013 International Conference on**. [S.l.: s.n.], 2013. p. 371–376.

RUDOWSKY, I. Intelligent agents. **The Communications of the Association for Information Systems**, v. 14, n. 1, p. 48, 2004.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. Second. [S.l.]: Prentice-Hall, Englewood Cliffs, NJ, USA, 2003.

SAYED, A. Adaptive networks. **Proceedings of the IEEE**, v. 102, n. 4, p. 460–497, April 2014. ISSN 0018-9219.

SCHLECHT, J. et al. Decentralized search by unmanned air vehicles using local communication. In: **IC-AI**. [S.l.: s.n.], 2003. p. 757–762.

SCHNEIDER, J.; NAGGATZ, M.; SPALLEK, R. Implementation of architecture concepts for hardware agent systems. In: **Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on**. [S.l.: s.n.], 2007. p. 823–828.

SCHUTZ, D. et al. Highly reconfigurable production systems controlled by real-time agents. In: **Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on**. [S.l.: s.n.], 2011. p. 1–8. ISSN 1946-0740.

SURESH, T.; HENRY, J.; RANGARAJAN, P. Multi-agent based reconfigurable architecture for future wireless networks. In: **Intelligent Agent Multi-Agent Systems, 2009. IAMA 2009. International Conference on**. [S.l.: s.n.], 2009. p. 1–5.

SYSTEMS, W. R. **Linux In Embedded Systems: Where Are The Benefits?** 2002.

TILAB. **<http://jade.tilab.com/>**. 2014. <http://jade.tilab.com/>.

URE, N. et al. **An Automated Battery Management System to Enable Persistent Missions With Multiple Aerial Vehicles**. 2014. 1-12 p.

VINYALS, M.; RODRIGUEZ-AGUILAR, J. A.; CERQUIDES, J. A survey on sensor networks from a multiagent perspective. **Comput. J.**, Oxford University Press, Oxford, UK, v. 54, n. 3, p. 455–470, mar. 2011. ISSN 0010-4620.

VRBA, P.; MARIK, V. Capabilities of dynamic reconfiguration of multiagent-based industrial control systems. **Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on**, v. 40, n. 2, p. 213–223, March 2010. ISSN 1083-4427.

VRBA, P. et al. **A Review of Agent and Service-oriented Concepts applied to Intelligent Energy Systems**. 2014. 1-1 p.

VYROUBAL, V.; KUSEK, M. Task migration of jade agents on android platform. In: **Telecommunications (ConTEL), 2013 12th International Conference on**. [S.l.: s.n.], 2013. p. 123–130.

WILLIAMS, J. W.; BERGMANN, N. W. Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. In: **ERSA**. [S.l.: s.n.], 2004. p. 163–169.

WOEBBECKE, D. et al. Color indices for weed identification under various soil, residue, and lighting conditions. **Transactions of the ASAE**, American Society of Agricultural Engineers, v. 38, n. 1, p. 259–269, 1995.

WOOLDRIDGE, M. **Introduction to MultiAgent Systems**. [S.l.]: John Wiley and Sons, 2002. ISBN 047149691X.

XILINX, I. **AXI Reference Guide UG761 (v13.1)**. 2011.

XILINX, I. **ML605 Hardware User Guide**. 2012.

XILINX, I. **Spartan-3E FPGA Family Data Sheet DS312 (v4.1)** . 2013.

XILINX, I. **Virtex-6 Family Overview**. 2015.

Xilinx, Inc. **Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices**. 2013.

YANG, H. et al. Review of advanced fpga architectures and technologies. **Journal of Electronics (China)**, v. 31, n. 5, p. 371–393, 2014. ISSN 1993-0615. Disponível em: <http://dx.doi.org/10.1007/s11767-014-4090-x>.

ZHANG, C.; KOVACS, J. M. The application of small unmanned aerial systems for precision agriculture: a review. **Precision agriculture**, Springer, v. 13, n. 6, p. 693–712, 2012.