Um framework para coprojeto de hardware e software de sistemas avançados de assistência ao motorista baseados em câmeras

Leandro Andrade Martinez

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Leandro Andrade Martinez

Um framework para coprojeto de hardware e software de
sistemas avançados de assistência ao motorista baseados
em câmeras

Tese apresentada ao Instituto de Ciências
Matemáticas e de Computação – ICMC-USP,
como parte dos requisitos para obtenção do título
de Doutor em Ciências – Ciências de Computação e
Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientador: Prof. Dr. Eduardo Marques

USP – São Carlos
Agosto de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

M375f Martinez, Leandro Andrade
Um framework para coprojeto de hardware e software de sistemas avançados de assistência ao motorista baseados em câmeras / Leandro Andrade Martinez; orientador Eduardo Marques. -- São Carlos, 2017.
130 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2017.

1. Coprojeto de Hardware e Software. 2. ADAS. 3. Sistemas Embarcados. 4. DSE. I. Marques, Eduardo, orient. II. Título.

Leandro Andrade Martinez

Hardware and software codesign framework for
camera-based advanced driver assistance systems

Doctoral dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Doctorate Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Eduardo Marques

USP – São Carlos
August 2017

*Dedico este trabalho aos meus pais Antonio Sérgio (in memoriam) e Maria Célia,
com todo meu amor e gratidão por tudo que me proporcionaram ao longo da vida.
Desejo ser merecedor dessa dedicação incansável, especialmente quanto à minha formação.*

AGRADECIMENTOS

Meu agradecimento especial ao meu grande amigo, prof. Dr. Eduardo Marques, que com toda sua dedicação, me deu todo apoio moral e intelectual, estando ao meu lado nos momentos mais difíceis dessa empreitada.

Agradeço também a CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) ¹ pelo importante apoio financeiro concedido à elaboração desta tese por meio de bolsa de estudos.

¹ <<http://www.capes.gov.br/>>

“Quem se arrisca a andar por ares nunca antes respirados, ou pensar fora da curva tem grandes chances de encontrar pedras no caminho. No entanto, ninguém é digno de contribuir para a ciência se não usar suas dores e insônias nesse processo. Não há céu sem tempestade. Risos e lágrimas, sucessos e fracassos, aplausos e vaias fazem parte do currículo de cada ser humano, em especial daqueles que são apaixonados por produzir novas ideias.”

(Dr. Augusto Cury, Ph.D.)

RESUMO

MARTINEZ, L. A. **Um framework para coprojeto de hardware e software de sistemas avançados de assistência ao motorista baseados em câmeras.** 2017. 130 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

A demanda por novas tecnologias, melhoria de segurança e conforto para veículos urbanos cresceu consideravelmente nos últimos anos, motivando a indústria na criação de sistemas destinados ao apoio de motoristas (ADAS - Advanced Driver Assistance Systems). Este fato contribuiu para o desenvolvimento de diversos sistemas embarcados na área automobilística destacando-se, à prevenção de colisão a pedestres por veículos. Através do avanço em diversas pesquisas, começaram a circular pelas ruas veículos com sistemas anticolisão e com navegação autônoma. Contudo, para alcançar objetivos cada vez mais desafiadores, os projetistas precisam de ferramentas que permitam unir tecnologias e conhecimentos de áreas distintas de forma eficiente. Nesse contexto, há uma demanda para a construção de sistemas que aumentem o nível de abstração da modelagem de projetos para o processamento de imagens em sistemas embarcados e assim, possibilitando uma melhor exploração do espaço de projetos. A fim de contribuir para minimizar este problema, este trabalho de pesquisa demonstra o desenvolvimento de um framework para coprojeto de hardware e software específico para a construção de sistemas ADAS que utilizam visão computacional. O Framework visa facilitar o desenvolvimento dessas aplicações permitindo a exploração o espaço de projeto (DSE - Design Space Exploration), e assim contribuindo para um ganho de desempenho no desenvolvimento de sistemas embarcados quando comparados à construção totalmente de um modo manual. Uma das características deste projeto é a possibilidade da simulação da aplicação antes da síntese em um sistema reconfigurável. Os principais desafios deste sistema foram relacionados à construção do sistema de intercomunicação entre os diversos blocos de Propriedade Intelectual (IP) e os componentes de software, abstraindo do usuário final inúmeros detalhes de hardware, tais como gerenciamento de memória, interrupções, cache, tipos de dados (ponto flutuante, ponto fixo, inteiros) e etc, possibilitando um sistema mais amigável ao projetista.

Palavras-chave: CoProjeto, ADAS, Sistemas Embarcados, Hardware.

ABSTRACT

MARTINEZ, L. A. **Hardware and software codesign framework for camera-based advanced driver assistance systems**. 2017. 130 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

The demand for new technologies, enhanced security and comfort for urban cars has grown considerably in recent years prompting the industry to create systems designed to support drivers (ADAS - Advanced Driver Assistance Systems). This fact contributed to the development of many embedded systems in the automotive area among them, the pedestrians collision avoidance. Through the advancement in various research, began circulating through the streets vehicles with anti-collision systems and autonomous navigation. However, to achieve ever more challenging goals, designers need tools to unite technology and expertise from different areas efficiently. In this context, there is a demand for building systems that increase the level of abstraction of models of image processing for use in embedded systems enabling better design space exploration. To help minimize this problem, this research demonstrates a develop a specific framework for hardware/software codesign to build ADAS systems using computer vision. The framework aims to facilitate the development of applications, allowing better explore the design space, and thus contribute to a performance gain in the development of embedded systems in relation to building entirely in hardware. One of the requirements of the project is the possibility of the simulation of an application before synthesis on a reconfigurable system. The main challenges of this system were related to the construction of the intercommunication system between the various Intellectual Property (IP) blocks and the software components, abstracting from the end user numerous hardware details, such as memory management, interruptions, cache, types (Floating point, fixed point, integers) and so on, enabling a more user-friendly system for the designer.

Keywords: Codesign, ADAS, Embedded Systems, Hardware.

LISTA DE ILUSTRAÇÕES

Figura 1 – Dados do Ministério da Saúde	27
Figura 2 – Estrutura interna de um FPGA	33
Figura 3 – Exploração do espaço de projetos	35
Figura 4 – Sistemas de Auxílio ao Condutor	37
Figura 5 – Visão geral do sistema	39
Figura 6 – Visão geral de um sistema de detecção de pedestres	39
Figura 7 – Distribuição de velocidade de impacto em acidentes com pedestres	40
Figura 8 – HLS Bluespec	41
Figura 9 – Processo de compilação Bluespec	42
Figura 10 – Varredura utilizada em filros de imagem	43
Figura 11 – Configurações de buffers de linha	44
Figura 12 – Captação de dados pela câmera D5M	44
Figura 13 – Padrão de recepção de imagem pela câmera D5M	45
Figura 14 – Características de exploração de ferramentas DSE	48
Figura 15 – Baselabs IDE	50
Figura 16 – Baselabs Aplicação	51
Figura 17 – Intempora IDE	52
Figura 18 – LabView IDE	53
Figura 19 – ImprovCV IDE	54
Figura 20 – MatLab - IDE para HDL	55
Figura 21 – MatLab - Exemplo de Aplicação	56
Figura 22 – Mescal	56
Figura 23 – Características de exploração de ferramentas DSE	59
Figura 24 – ADAS-Vision Framework	61
Figura 25 – Componentes da placa de desenvolvimento DE2i-150	62
Figura 26 – Tipos de bibliotecas de componentes	64
Figura 27 – Processo de DSE na ferramenta	65
Figura 28 – Fluxo geral de funcionamento do sistema proposto	66
Figura 29 – Esboço da Interface da aplicação	67
Figura 30 – Esboço de um IP no sistema proposto	68
Figura 31 – Esboço de um projeto esquemático no sistema proposto	69
Figura 32 – Diagrama de Classes do projeto	70
Figura 33 – Diagrama de Sequencia de um IP Software	71

Figura 34 – Diagrama de Sequencia de um IP Hardware	73
Figura 35 – Funcionamento do Controlado de Bancada	74
Figura 36 – Simulação de um IP em Hardware	77
Figura 37 – Customização de IP em tempo de execução	78
Figura 38 – Co-Simulação com pré-processamento	79
Figura 39 – Co-Simulação usando FPGA como dispositivo (<i>runtime</i>)	80
Figura 40 – Co-Simulação usando barramento Avalon no FPGA	81
Figura 41 – Co-Simulação usando NIOS II	82
Figura 42 – Co-Simulação usando NIOS II com instrução customizada	82
Figura 43 – Fluxo completo para compilação de um projeto em Hardware	84
Figura 44 – Construção de IPs	85
Figura 45 – Interligação entre IPs <i>glue-box</i>	86
Figura 46 – Diversas apresentações de um IP	87
Figura 47 – Validação de um IP em Software	88
Figura 48 – Validação de um IP em Hardware	88
Figura 49 – IPs para ADAS-VISION	92
Figura 50 – Processo de exploração do espaço de projetos	95
Figura 51 – Estudo de caso: Projeto 1	96
Figura 52 – Projeto 1 em simulação na ferramenta	98
Figura 53 – Resultado comparativo: Projeto 2	99
Figura 54 – Estudo de caso: Projeto 2	100
Figura 55 – Projeto 2 em simulação na ferramenta	102
Figura 56 – Projeto 2 análise de dependências	103
Figura 57 – Planilha para tomada de decisão	105
Figura 58 – Arvore de decisão	107
Figura 59 – Gráfico de Pareto	108
Figura 60 – Formação dos Diretórios do Sistema	127

LISTA DE CÓDIGOS-FONTE

Código-fonte 1	– Subrotina de um bloco de controlador de bancada em Hardware . . .	75
Código-fonte 2	– Subrotina de um bloco IP em Hardware	76
Código-fonte 3	– Subrotina de um bloco IP compilado para Verilog	119
Código-fonte 4	– Subrotina de teste de bancada compilado em verilog	121
Código-fonte 5	– Subrotina de arquivo em lote para compilação pelo simulador	126
Código-fonte 6	– Subrotina de arquivo em lote para execução da simulação	126

LISTA DE TABELAS

Tabela 1 – Análise combinatória para diversos modelos de projetos	91
Tabela 2 – Tabela de atributos para Modelo Qualidade (QM) para Sistemas Embarcados Críticos (CES)	94

LISTA DE ABREVIATURAS E SIGLAS

ACC	<i>Adaptative Cruise Control</i>
ADAS	<i>Advanced driver assistance systems</i>
AMBA	<i>Advanced Microcontroller Bus Architecture</i>
ANN	<i>Artificial neural networks</i>
ASIC	<i>Application Specific Integrated Circuits</i>
ASSP	<i>Application-specific standard parts</i>
BGA	<i>Ball Grid Array</i>
BLOB	<i>Binary Large Object</i>
CCD	<i>Charge-Coupled Device</i>
CES	<i>Critical Embedded Systems</i>
CI	<i>Circuito Integrado</i>
CMOS	<i>Complementary Metal-Oxide-Semiconductor</i>
COM	<i>Component Object Model</i>
CPU	<i>Central Processing Unit</i>
CSDF	<i>Cyclo-Static Data Flow</i>
DPLA	<i>Dynamic Programmable Logic Array</i>
DPRF	Departamento de Polícia Rodoviária Federal
DSE	<i>Design Space Exploration</i>
DSL	<i>Domain-Specific Languages</i>
DSP	<i>Digital Signal Processor</i>
E/S	<i>Entrada e Saída</i>
ePROM	<i>Erasable Programmable Read-Only Memory</i>
ESL	<i>Electronic System-Level</i>
FIFO	<i>First In, First Out</i>
FPGA	<i>Field Programmable Gate Array</i>
FPLA	<i>Field Programmable Logic Array</i>
FSM	<i>Finite State Machine</i>
GPU	<i>Graphics Processing Unit</i>
HDL	<i>Hardware description language</i>
HLS	<i>High-level synthesis</i>
HoG	<i>Histogram of Oriented Gradients</i>

HWD	<i>Hardwired logic devices</i>
IDE	<i>Integrated Development Environment</i>
IP	<i>Intellectual property</i>
ISA	<i>Intelligent Speed Advice</i>
ISP	<i>Image Signal Processor</i>
KNN	<i>K-Nearest Neighbors</i>
KPN	<i>Kahn Process Networks</i>
LBP	<i>Local binary patterns</i>
LCD	<i>Liquid Crystal Display</i>
LCR	<i>Laboratório de Computação Reconfigurável</i>
LE	<i>Logic Elements</i>
MDE	<i>Model-Driven Engineering</i>
MoA	<i>Modelo de Arquitetura</i>
MoC	<i>Modelo de Computação</i>
OBDDs	<i>Ordered Binary Decision Diagrams</i>
PbD	<i>Platform-based Design</i>
PCIe	<i>Peripheral Component Interconnect-Express</i>
PET	<i>Parametric Exploration Tool</i>
PLD	<i>Programmable Logic Devices</i>
PLL	<i>Phase-Locked Loop</i>
PPS	<i>Pedestrian Protection System</i>
QM	<i>Quality Model</i>
RAM	<i>Random Access Memory</i>
RGB	<i>Red Green Blue</i>
ROI	<i>Region Of Interest</i>
RTL	<i>Register Transfer Level</i>
SADF	<i>Scenario-Aware Data Flow</i>
SDF	<i>Synchronous Data Flow</i>
SDRAM	<i>Synchronous Dynamic Random Access Memory</i>
SIFT	<i>Scale-invariant feature transform</i>
SLD	<i>System-Level Design</i>
SoC	<i>System-on-Chip</i>
SoPC	<i>System on Programmable Chip</i>
SVM	<i>Support Vector Machine</i>
VGA	<i>Video Graphics Array</i>
VHDL	<i>VHSIC Hardware Description Language</i>
VHSIC	<i>Very High Speed Integrated Circuits</i>

SUMÁRIO

1	INTRODUÇÃO	25
1.1	Justificativa	25
1.2	Motivação	26
1.3	Objetivos	28
1.4	Organização do Trabalho	29
2	FUNDAMENTAÇÃO TEÓRICA	31
2.1	Computação Reconfigurável	31
2.1.1	<i>Field Programmable Gate Array (FPGA)</i>	32
2.1.2	<i>Electronic System-Level (ESL)</i>	33
2.2	Coprojeto Hardware / Software	34
2.3	Co-Simulação	35
2.4	Aplicações para ADAS	36
2.5	Linguagens de descrição de hardware	40
2.5.1	<i>Verilog e SystemVerilog</i>	40
2.5.2	<i>Bluespec SystemVerilog</i>	41
2.6	Processamento de Imagem	42
2.6.1	<i>Filtros de Convolução</i>	43
2.6.2	<i>Módulos de captura</i>	44
3	ESTADO DA ARTE	47
3.1	Ferramentas para Exploração do Espaço de Projetos (DSE)	48
4	IMPLEMENTAÇÃO DO PROJETO	61
4.1	Plataforma de Hardware	62
4.2	Construção do Software	63
4.2.1	<i>Especificação da Interface Gráfica</i>	67
4.2.2	<i>Especificação do Projeto de Software</i>	69
4.3	Simulador de coprojetos	73
4.3.1	<i>Customização de IP em tempo de Execução</i>	76
4.4	Gerador de Projetos	85
4.4.1	<i>Validações de Núcleos de Propriedade Intelectual</i>	86
5	RESULTADOS	89

5.1	Exploração do Espaço de Projetos (DSE)	90
5.1.1	<i>Método de Exploração do Espaço de Projetos</i>	93
5.2	Resultados do Primeiro Projeto	93
5.3	Resultados do Segundo Projeto	99
5.4	Discussão sobre os resultados obtidos	104
6	CONCLUSÕES	109
6.1	Aspectos Gerais	109
6.2	Contribuições do Trabalho	109
6.3	Trabalhos Futuros	111
REFERÊNCIAS		113
APÊNDICE A DOCUMENTAÇÃO DOS BLOCOS DE HARDWARE .		119
A.1	Exemplo Código Fonte de um IP	119
ANEXO A SITES RELACIONADOS AO PROJETO		129
A.0.1	<i>Bibliotecas e Softwares utilizados</i>	129

INTRODUÇÃO

A necessidade da redução do número de acidentes de trânsito concentra esforços de cientistas em todo o mundo, pois, grande parte dos acidentes são decorrentes da ineficiência do cérebro humano em responder rapidamente à situações de perigo ou prever situações de alto risco. Assim, existe atualmente uma demanda considerável para soluções tecnológicas direcionadas a auxiliar os sistemas ADAS.

A computação reconfigurável se destaca como uma nova e importante estrutura para computação embarcada combinando programabilidade com a capacidade de efetuar computação de propósito específico e eficiente. Dessa forma, este projeto se enquadra em estabelecer uma proposta para viabilizar a implementação de um *framework* que possibilite a exploração do espaço de projeto e diminuição do esforço de desenvolvimento, visando um ganho na velocidade de produção de sistemas ADAS baseados em visão. Como resultado, o primeiro caso de teste do *framework* é um sistema embarcado de predição de colisão a pedestres baseado no coprojeto hardware e software, podendo contribuir futuramente para a diminuição do número de acidentes de trânsito. Os principais desafios deste sistema foram relacionados à construção do sistema de intercomunicação entre os diversos blocos de Propriedade Intelectual (IP) e os componentes de software, abstraindo do usuário final inúmeros detalhes de hardware, tais como gerenciamento de memória (*on-chip*, *off-chip*), interrupções, cache, tipos de dados (ponto flutuante, ponto fixo, inteiros) e etc.

1.1 Justificativa

Devido ao fato das linguagens de descrição de hardware (HDL) trabalharem em baixo nível na codificação, um tempo significativo do ciclo de projeto é gasto na programação e na verificação. As iterações de projeto aumentam os custos podendo se tornar proibitivas para uma minuciosa exploração do espaço de projetos. Assim, há uma demanda para melhorar o aspecto de programação de FPGAs para se adequar ao corrente fluxo de projetos ligados a computação

heterogênea. Nesse contexto, os *frameworks* baseados em modelo fornecem uma forma abstrata de projetar controles complexos e processamento de sinais, melhorando a qualidade e acelerando tarefas de projeto e verificação (BACON; RABBAH; SHUKLA, 2013).

No começo da década, uma pesquisa de projeção previu que a taxa de reuso em circuitos lógicos na indústria deveria crescer linearmente. Em 2011 essa taxa era de 54% e deverá chegar a 98% em 2026. O que confirma esta tendência são os recentes circuitos FPGA que hoje atingem a faixa aproximada de 72 milhões de portas lógicas. Com esses dados, confirma-se a necessidade de melhores ferramentas para projetos *System-on-Chip* (SoC) para atender a demanda de produtividade. Para resolver este desafio, várias abordagens devem ser combinadas. Em primeiro lugar, os níveis de abstração de projetos devem ser elevados. Em segundo lugar, deve ser aumentado o grau de automação, particularmente na verificação e implementação de projetos (ITRS, 2011; GROUP, 2017).

Conforme descrito por Bacon, Rabbah e Shukla (2013), há uma tendência para para integração de CPUs, GPUs, FPGAs e outros ASSPs em um mesmo chip. Existem padrões e especificações bem estabelecidos para co-programação de CPUs e GPUs, contudo há uma lacuna para integração com FPGA justificada pela falta de *drivers* de dispositivos, linguagens de programação e ferramentas, o que contribui como argumento para elaboração deste trabalho.

1.2 Motivação

Segundo dados obtidos pelo relatório de **Número de Acidentes por Gravidade** (DPRF, 2011, p. 28) do Departamento de Polícia Rodoviária Federal (DPRF), ocorreram em torno de 189 mil acidentes de trânsito (em rodovias) com aproximadamente 7 mil vítimas fatais. Em outra base, as informações do Ministério da Saúde (DATASUS, 2015) complementa que, nos diversos tipos de acidente, apenas no ano de 2014, ocorreram cerca de 43 mil mortes relacionadas aos acidentes de trânsito. A Figura 1 mostra a evolução do número de óbitos registrados entre 2003 a 2015, mostrando uma tendência crescente no número de óbitos. Em consequência, gera-se uma significativa quantidade de despesas médicas e sociais ao País, além dos prejuízos causados às famílias das vítimas.

Um outro aspecto a ser considerado é que os carros modernos estão vindo de fábrica cada vez mais com os sistemas ADAS e isso tem gerado um falso sentimento de confiabilidade ao usuário durante o processo de condução do veículo. Recentemente, as mídias digitais especializadas ¹ em sistemas embarcados divulgaram um acidente ocorrido nos Estados Unidos em maio deste ano com um veículo de última geração (modelo Tesla S), onde uma falha grave no sistema do piloto automático causou a morte do motorista. O acidente despertou novamente a questão da confiabilidade de tais sistemas ADAS e muitas perguntas sobre o ocorrido estão sem

¹ Referência proveniente da revista EETimes: Tesla's Fatal Crash: 6 Unanswered Questions (YOSHIDA, 2016).

respostas:

1. Que parte do sistema (sensor CMOS de imagem, radar, processador de visão) do piloto automático do Tesla não reconheceu o perigo iminente da colisão com outro veículo?
2. Como o sistema de piloto automático da Tesla é projetado para interpretar dados sensoriais?
3. Qual hardware foi responsável pela fusão de sensores?
4. Quem escreveu os algoritmos de fusão de sensores determinando quais dados se sobressaem em relação a outros, e o que acontece quando diferentes sensores fornecem informações contraditórias?
5. Mais especificamente, como o Tesla integrou o processador de visão em seu sistema de emergência automática?

Todas essas questões demonstram que os sistemas ADAS ainda estão longe de ser uma tecnologia madura e estabilizada.

Figura 1 – Dados do Ministério da Saúde

Fonte: DATASUS (2015, online).

Levando-se este contexto em consideração, a motivação deste projeto resume-se a:

1. Construção de uma ferramenta didática para o ensino de disciplinas na área de Sistemas Embarcados, que permita a exploração do espaço de projetos (DSE)
2. Criar um framework que facilite a construção de aplicações de ADAS e, que por consequência, possa contribuir em pesquisas que colaborem com a diminuição do crescente número de vítimas de trânsito.
3. Trazer o conhecimento de aplicações da visão robótica aplicada aos sistemas embarcados, com estudos de tecnologias e que possam contribuir para a construção de sistemas autônomos direcionados a detecção de colisão.

4. Há muitos casos (como o caso Tesla) que são quase impossíveis de testar. Considerando o número infinito de cenários potenciais que poderiam levar a um acidente, e como a indústria automobilística planeja enfrentar o desafio de modelagem, simulação, teste e validação, evidencia-se a necessidade de uma ferramenta que explore o espaço de projeto testando a maior quantidade possível de cenários e situações críticas visando uma confiabilidade maior no produto.

Na área da Engenharia de Software, a reutilização de sub-rotinas de código já é um processo comum e utilizado há bastante tempo. Recentemente, houve um aumento de diversas entidades como Altera, Easic, OpenCores, Mentor Graphics, Xilinx, Synopsys entre outras, para o reuso também de hardware. Projetos como dos autores de [Heish e Lin \(2013\)](#), já implementados com esses IPs reutilizáveis, mostraram-se tão eficientes quanto versões disponíveis pelo mercado. Muitos desses IPs de Hardware são disponibilizados gratuitamente para uso não comercial, dessa forma, diversos novos IPs podem ser incorporados ao *framework* do ADAS-Vision com objetivo de expandir a capacidade da exploração do espaço de projetos.

1.3 Objetivos

O objetivo principal desta pesquisa foi construir um framework para o coprojeto de hardware e software de sistemas ADAS baseados em visão, visando a exploração do espaço de projetos das aplicações automotivas.

E os objetivos secundários foram:

- Desenvolvimento de sistemas embarcados de hardware/software para aplicações ADAS com desempenho em Tempo Real.
- Desenvolvimento de sistemas embarcados de hardware/software para aplicações ADAS baseado em Computação Reconfigurável, visando explorar ao máximo a flexibilidade das aplicações;
- Construção de diversos blocos de IPs em Hardware (especificados na HLS Bluespec) tais como: filtro de ruídos, detecção de movimento, Blob, estabilizador de vídeo, classificador de pedestres, rastreamento, Super-Resolução, interfaces de E/S, teste e depuração, alocação de memória, etc.
- Customização dos diversos blocos de IPs em Hardware, explorando a área (elementos lógicos), consumo energético e velocidade.
- Construção de diversos componentes de Software (especificados em linguagem C) tais como: filtro de ruídos, detecção de movimento, Blob, estabilizador de vídeo, classificador de pedestres, tracking, Super Resolução, interfaces de E/S, teste e depuração, etc.

- Construção e Integração de processador soft-core (NIOS II) para a execução de componentes de software embarcados (SoC).
- Geração automática de scripts para a instanciação das diversas ferramentas (Bluespec, BlueSim, Quartus II, Eclipse IDE, QSYS, OpenCV, etc) associadas ao *framework*.
- Construção de gabaritos (*templates*) para as diversas aplicações ADAS com visão (Detecção de pedestres, Ponto cego, detecção de mudança de faixa, análise de placas de trânsito, etc), em diversas condições climáticas (chuva, neblina, ensolarado, noite)

1.4 Organização do Trabalho

A seguir, uma breve descrição da organização da tese:

Capítulo 1 - Introdução: São apresentados os objetivos a fim de situar o leitor no domínio do problema abordado. O conteúdo apresenta a motivação na qual o trabalho está fundamentado, assim como, o contexto em que se situa, as áreas onde se aplica e por fim os principais desafios e propósitos necessários para a execução do projeto.

Capítulo 2 - Fundamentação Teórica: São apresentados os métodos e técnicas utilizados no desenvolvimento do projeto. Há também um breve histórico dos métodos aplicados.

Capítulo 3 - Estado da Arte: Principais sistemas relacionados com este trabalho de pesquisa.

Capítulo 4 - Desenvolvimento: São apresentadas informações a respeito das tecnologias e métodos disponíveis para o desenvolvimento do projeto proposto.

Capítulo 5 - Resultados: Neste capítulo há um detalhamento do projeto proposto assim como um escopo da utilização do Framework.

Capítulo 6 - Conclusões: No capítulo final, há uma descrição mostrando como o *framework* desenvolvido contribuiu para alcançar os objetivos que foram propostos. Também são apresentadas as contribuições que este trabalho oferece e por fim, sugestões para trabalhos futuros.

FUNDAMENTAÇÃO TEÓRICA

O objetivo deste capítulo é apresentar uma visão geral de tecnologias envolvidas com este trabalho, principalmente relacionados aos Sistemas Embarcados e a Computação Reconfigurável. Na [seção 2.1](#) é apresentado um breve resumo sobre a Computação Reconfigurável, que é a área principal de atuação deste projeto de pesquisa. Na [seção 2.2](#) são apresentados os conceitos básicos da construção de sistemas de hardware e software integrados. Na [seção 2.3](#) está descrito os princípios envolvidos na construção de simuladores simultâneos de hardware e software. Na [seção 2.4](#) há uma breve referência aos ADAS que são sistemas alvo de modelagem pelo *framework* proposto. Na [seção 2.5](#) são apresentadas algumas características importantes de algumas Linguagens de Descrição de Hardware (HDLs), essas características foram determinantes para que essas linguagens fossem escolhidas para uso neste projeto. E, ao final, na [seção 2.6](#) há um resumo das principais técnicas envolvidas com a construção de blocos de hardware para validação da ferramenta.

2.1 Computação Reconfigurável

Uma das principais motivações para a pesquisa e desenvolvimento de computadores paralelos é o problema chamado “*gargalo de von Neumann*”. Este problema ocorre porque a velocidade da conexão entre a memória e o processador geralmente limita a velocidade do computador, pois em muitos casos, as instruções podem ser executadas mais rapidamente do que sua transferência para o processador executar. Nesse aspecto, a programação paralela pode incrementar o desempenho em relação à regra de uma instrução de cada vez por processador. Os circuitos eletrônicos destacam-se nessa resolução pois permitem que algoritmos sejam prototipados e embarcados ([SEBESTA, 2012](#), p. 27).

A computação reconfigurável pode ser definida como o estudo da computação envolvendo dispositivos reconfiguráveis ([BOBDA, 2007](#)). Estes dispositivos tem a capacidade de re-programação de hardware, onde é possível adaptar sua utilização em determinados momentos

(COMPTON; HAUCK, 2002). Nesse contexto, incluem-se também: as aplicações, as arquiteturas e os algoritmos relacionados.

Segundo (CARDOSO; HÜBNER, 2011), uma aplicação clássica para os subsistemas computacionais reconfiguráveis é sua utilização como um acelerador de apoio a CPU (Unidade de Processamento Central). Os aceleradores (que normalmente não são baseados na arquitetura de *von-Neumann*) podem ser divididos em dois grupos: (1) dispositivos lógicos com fio - *Hardwired logic devices* (HWD) e (2) dispositivos lógicos programáveis - *Programmable Logic Devices* (PLD), onde o termo “programável” indica a funcionalidade da reconfiguração. A fabricação dos blocos programáveis tiveram seu início na década de 1980 onde foram chamados de *Field Programmable Logic Array* (FPLA) e posteriormente de *Dynamic Programmable Logic Array* (DPLA), apresentando um arranjo semelhante à memória ePROM (memória programável apagável somente de leitura) e onde era possível apenas fazer a soma de produtos através de funções booleanas. Mesmo assim, representou um fator de aumento de velocidade de 15000 vezes quando usado para combinar centenas de expressões booleanas dentro de um único ciclo de relógio, ao invés de calcular sequencialmente por um microprocessador. Através do esforço de várias universidades e organizações voltadas para o desenvolvimento de chips, foi fabricado um DPLA contendo um conjunto dos primeiros 256 FPGAs e que só foram lançados ao mercado em 1984 pela empresa *Xilinx*.

Com o advento da reconfiguração, muitos núcleos em hardware de propriedade intelectual (*hardware IP cores*) tornaram-se disponíveis, e diversas iniciativas como o opencores.org¹ que disponibilizam hardware com código fonte aberto (*open source hardware*) pela internet. Estudos da indústria eletrônica comprovam que a produtividade pode dobrar quando há uma reutilização de softwares e IP de hardware (ITRS, 2011; HEISH; LIN, 2013).

2.1.1 *Field Programmable Gate Array (FPGA)*

De forma geral, quando analisado sob as métricas de flexibilidade e desempenho, a função do FPGA encontra-se no preenchimento da lacuna que existente entre o hardware e o software. Isso torna possível que o FPGA tenha maior flexibilidade de alterações quando comparado a um sistema totalmente em hardware e permite um ganho de desempenho quando comparado a aplicações totalmente em software.

Os dispositivos FPGA já estão consolidados no mercado como uma ferramenta de prototipagem de hardware. Processadores Pentium e Atom da Intel foram sintetizados inicialmente em FPGA (WANG *et al.*, 2009). A principal vantagem da construção de um sistema customizado usando FPGA é a possibilidade para rápidas modificações de projeto. Em desvantagem está o custo associado a estes dispositivos e sua baixa frequência de clock (KIRISCHIAN; GEURKOV; KIRISCHIAN, 2008). Entretanto, o principal desafio está na criatividade e a flexibilidade para reorganizar a aplicação e explorar o espaço de projeto de uma maneira eficiente. Atualmente,

¹ O site OpenCores.org pode ser acessado em: opencores.org (acessado em junho de 2017)

os protótipos e produtos ADAS que utilizam câmeras são baseados em dispositivos de alto desempenho através de Processadores Digitais de Sinais (DSPs), combinados com hardware reconfigurável com utilização de processamento paralelo (TECHMER, 2007).

Um FPGA consiste de um dispositivo programável composto por três partes principais: um conjunto de blocos lógicos, uma matriz programável de interconexões e um conjunto de células de entrada e saída em torno do dispositivo, como ilustrado na Figura 2.

Figura 2 – Estrutura interna de um FPGA

Fonte: Bailey (2011, Traduzido).

2.1.2 Electronic System-Level (ESL)

A necessidade de sistemas embarcados cada vez mais complexos e o avanço contínuo da capacidade dos circuitos integrados, estão direcionando o foco do desenvolvimento de sistemas digitais para ambientes com alto nível de abstração, e dessa forma, cada vez mais distantes do desenvolvimento de hardware em baixo nível (SHUKLA; PIXLEY; SMITH, 2006; COUSSY; MORAWIEC, 2008; COUSSY *et al.*, 2009). Os projetos de sistemas evoluíram passando do nível de portas lógicas (*Gates*), para o nível de registradores (RTL) e, atualmente, os projetos são direcionados para ferramentas de mais alto nível (ESL), tornando-se cada vez mais comum a utilização de paradigmas de desenvolvimento de nível de sistema (SLD) (DENSMORE; PASSERONE, 2006) (Martin e Bailey, 2007) (PATEL, 2008).

O ESL apresenta diversas vantagens em relação ao RTL, facilitando o coprojeto de hardware e software e proporcionando maior velocidade na simulação, possibilitando testes e

ajustes da arquitetura do sistema em alto nível ([DRECHSLER, 2004](#)).

2.2 Coprojeto Hardware / Software

A área de coprojeto de hardware/software surgiu na década de 90 basicamente como uma disciplina para projetar circuitos integrados complexos. Guiados pelas previsões de avanços tecnológicos de Gordon Moore, as técnicas de coprojeto tornaram-se bem sucedidas no projeto de sistemas eletrônicos e são usadas atualmente pelas empresas de desenvolvimento de sistemas eletrônicos embarcados.

A construção de projetos de hardware e software elaborados separadamente pode levar a implementações de sistemas que não satisfaçam todas as propriedades funcionais, tais como tempo de execução, custo ou consumo de energia. Alternativamente, as decisões de projeto para a alocação de recursos poderiam não estar otimizadas para a necessidade dos requisitos, levando a implementações de sistemas muito caras e, portanto, superestimadas e reduzindo posteriormente a lucratividade por unidade vendida. Com essa característica, a exploração do espaço de projetos (DSE) tornou-se um elemento distinto na área de coprojetos.

Conforme descrito por [Teich \(2012\)](#), o principal objetivo da DSE é explorar um conjunto de implementações eficientemente viáveis e encontrando soluções ótimas. A [Figura 3](#) mostra o funcionamento de ferramentas para realizar a exploração de espaço de projetos baseados em modelos de aplicação ou modelos de computação (MoC) e modelo de arquitetura (plataforma) (MoA). Durante o DSE, o espaço de projeto dos candidatos a implementação são explorados. Tipicamente, cada síntese candidata é avaliada de acordo com definições implementadas por funções de avaliação.

Embora pesquisas significativas tenham ocorrido no desenvolvimento de técnicas de coprojeto de hardware/software nos últimos 25 anos, a grande parte dos projetistas de FPGA ainda seleciona manualmente a computação em sistemas heterogêneos que será mapeada para os recursos reconfiguráveis. À medida que os projetistas caminham para especificações mais comportamentais essa situação pode mudar ([TESSIER; POCEK; DEHON, 2015](#)).

Uma vantagem significativa do uso de FPGAs é a capacidade de especialização de um circuito lógico criado pelo projetista. Um hardware com parâmetros relacionados à uma função específica pode ser intensamente reduzido caso seja possível previamente selecionar as entradas dos parâmetros ao invés de programar a lógica da função. Em software é muito comum chamar uma função para calcular um valor e usá-lo em uma lógica, contudo, se essa função for usada com valores constantes não é necessário implementar um módulo para cálculo desta função em hardware, basta usar o valor previamente calculado (fixo) como parâmetro.

Embora as poderosas ferramentas de síntese de lógica FPGA aproveitem a especialização dependente de dados, ainda há uma lacuna no desenvolvimento de interfaces padronizadas para

Figura 3 – Exploração do espaço de projetos

Fonte: Teich (2012, Traduzido).

computação especializada em tempo de execução ² quando constantes são especificadas pelo usuário em tempo de compilação.

2.3 Co-Simulação

Um dos grandes desafios em projetar sistemas que funcionam em ambientes reais é mesclar as interações que existem entre softwares, plataformas, sistemas de comunicações e componentes físicos. O sistema de controle automotivo é um exemplo típico e muitas vezes projetado com base em um paradigma restrito a tempo de resposta. Uma das formas para contribuir com este problema é a criação de *frameworks* que permitam a exploração do espaço de projetos (DSE). Neste caso, a simulação feita através de diferentes tipos de sistemas como, por exemplo, os simuladores de hardware automotivos e os simuladores de ambientes físicos, fornecem importantes resultados para validações de projetos (ZHANG *et al.*, 2013).

Em contrapartida, verificação de circuitos eletrônicos modernos tornou-se o gargalo no tempo de projeto de um SoC. Um dos problemas está relacionado a detecção de erros de

² Neste caso, o termo *tempo de execução* ou *runtime* (termo em inglês), refere-se ao período em que o circuito já sintetizado está em execução no FPGA. Em contraponto, o termo *tempo de compilação*, refere-se ao período em que o código está sendo sintetizado para gerar um arquivo que posteriormente possa ser carregado pelo FPGA.

programação (*bugs*) que podem ser minimizados através da co-simulação. O projeto de [Campbell et al. \(2016\)](#), permite detectar e auxiliar na localização de erros lógicos de especificações C / C++ e também no mecanismo de síntese de alto nível. A estrutura de co-simulação de hardware/software permite depurar e verificar um projeto de SoC.

2.4 Aplicações para ADAS

Os sistemas de controle automotivos (ADAS), estão se tornando mais complexos e predominantes na indústria automotiva. Diversas aplicações são críticas, como por exemplo prevenção de colisão de emergência, controle de cruzeiro adaptativo, auto-estacionamento, advertência da mudança de pista, o planejamento da rota, etc. Dessa forma, há uma crescente demanda para uma metodologia de projeto e avaliação que seja eficiente para o desenvolvimento desses sistemas de controle automotivos ([KIM; SHIN, 2014](#)).

Na pesquisa de [Yamaura et al. \(2016\)](#), há um trabalho sobre uma estrutura de simulação em malha fechada que melhora o design e a avaliação de ADAS. Essa estrutura de simulação consiste em quatro ferramentas: Dymola (Simulador de modelos de dinâmica de veículos), Simulink (modelagem de software/hardware de controle de veículos), OpenMETA (integração ferramentas de projeto) e o Unity (mecanismo utilizado para jogos 3D). Nesse sistema, ferramentas PET (Ferramenta de Exploração Paramétrica) e DSE (Exploração de Espaço de Projetos) são fundamentais para melhorar a capacidade e a eficiência do projeto. O Unity tem funcionalidade chave para visualização e simulações de ADAS interativas ou em malha fechada, que contém modelos de sensores para ADAS, modelos de ambiente rodoviário.

Com a demanda crescente de Mercado, há uma grande variedade de sistemas ADAS, destacando-se: Adaptive cruise control (ACC); Lane departure warning system; Lane change assistance; Collision avoidance system (Precrash system); Intelligent speed adaptation or intelligent speed advice (ISA); Night Vision; Adaptive light control; Pedestrian protection system; Automatic parking; Traffic sign recognition; Blind spot detection; Driver drowsiness detection; Vehicular communication systems; Hill descent control;

A figura [Figura 4](#) mostra uma visão geral da área de Sistemas de Auxílio ao Condutor, em destaque os ADAS e que utilizam sistema de visão. Nota-se neste diagrama, que existem mais de 20 sistemas diferentes que utilizam câmeras, demonstrando-se assim a importância da pesquisa do contexto deste trabalho.

Figura 4 – Sistemas de Auxílio ao Condutor

Fonte: Zhao (2015, Adaptado).

Sistemas de proteção a pedestres

Por tratar-se de sistemas críticos, sistemas, processos e componentes envolvidos nos sistemas avançados para auxílio a motoristas devem ser seguros e confiáveis. Contudo, validações e testes exaustivos aumentam muito o investimento necessário acarretando um dos aspectos mais desafiadores do desenvolvimento desses sistemas. Tratando-se de sistemas de visão, faz-se necessário uma grande quantidade de bases de dados para testar todos os cenários e condições normais de trânsito para obter grande acurácia. Para isso, essas bases de vídeos devem ser reunidas e executadas em banco de dados para teste de regressão (KISAČANIN; GELAUTZ, 2014; BENGLER *et al.*, 2014).

De forma geral, os sistemas ADAS baseados em visão podem ter diferentes tipos de processamento paralelo. A Figura 5 mostra o princípio da cadeia de processamento de sistemas ADAS (TECHMER, 2007) relacionando níveis de complexidade. Os primeiros níveis desta cadeia estão relacionados ao tratamento intensivo dos dados provenientes da câmera. O segundo nível (médio) refere-se a descoberta de objetos na cena, e por fim, em um nível mais alto, a classificação e rastreamento. Durante a sequência, o nível de abstração aumenta a cada sequência.

Os Sistemas de Proteção a Pedestres (PPS) tem por objetivo, melhorar a segurança de tráfego. Nas ativas pesquisas da área, a detecção de pedestres é um dos principais processos. O fluxo de processamento baseado em módulos pode incluir seis módulos em cascata: pré-processamento, segmentação de primeiro plano, classificação de objetos, verificação / refinamento, rastreamento e aplicação (GERONIMO *et al.*, 2010).

Os três primeiros módulos podem ser considerados como processamento de nível baixo e médio e, portanto, concentrando maior interesse em otimizações, uma vez que uma implementação eficiente deles necessita de adequada arquitetura de hardware, sendo um dos pontos críticos do projeto (TECHMER, 2007).

As principais etapas envolvidas na arquitetura de um sistema de visão computacional são: Sensoriamento (Aquisição), Processamento de Imagens (Pré-processamento), Segmentação, Descrição, Reconhecimento e Interpretação. O gráfico da Figura 6 demonstra uma visão geral de um sistema de detecção de pedestres para uma câmera em movimento.

Uma das formas utilizadas para validar o framework neste projeto de doutorado consiste na execução de módulos desenvolvidos para o reconhecimento através de imagens em tempo real, em hardware reconfigurável e fazendo a detecção de pedestres em ambiente urbano real. O sistema foi projetado para detectar pedestres com veículos em velocidades de até 60 Km/h. Este limite de velocidade foi determinado baseando-se nos dados apresentados na Figura 7. Observa-se que após 60 km/h a taxa de acidentes tem pouca variação, contudo o esforço computacional para processar os dados com um veículo a 80 km/h cresce rapidamente. Neste caso, pretende-se avaliar técnicas de estabilização de imagem, sistemas de reconhecimento e sistemas de rastreamento a fim de combinar seus benefícios. O trabalho utilizará como base o trabalho de mestrado do

Figura 5 – Visão geral do sistema

Fonte: Martinez (2011).

próprio autor e trabalhos anteriores disponíveis por pesquisadores do ICMC.

Figura 6 – Visão geral de um sistema de detecção de pedestres

Fonte: Martinez (2011).

Figura 7 – Distribuição de velocidade de impacto em acidentes com pedestres

Fonte: Gavrilu e Munder (2006).

Bases de Dados disponíveis para detecção de pedestres

Para utilização no *framework* diversas bases de dados são compatíveis. A utilização dessas bases podem ser usadas para comparativo de resultados de trabalhos já publicados nas áreas de visão para sistemas ADAS. A seguir, alguns exemplos³:

- Caltech Pedestrian Detection Benchmark (CALTECH.EDU, 2017)
- Elektra Pedestrian Detection Datasets <<http://adas.cvc.uab.es/elektra/datasets/>>
- INRIA Person Dataset <<http://pascal.inrialpes.fr/data/human/>>
- Pascal Visual Object Challenges <<http://host.robots.ox.ac.uk/pascal/VOC/index.html>>
- Gavrilu Datasets <<http://www.gavrilu.net/Datasets/datasets.html>>

2.5 Linguagens de descrição de hardware

2.5.1 Verilog e SystemVerilog

O Verilog tem muitos recursos similares ao VHDL. Contudo, em contraste com VHDL, que tem sua origem na linguagem Ada, o Verilog é inspirado na modelagem da linguagem C. Consequentemente, é mais compacto, mas pela característica de linguagem de modelagem, nem todas as construções são sintetizáveis (IEEE, 2005). O SystemVerilog é uma extensão da linguagem Verilog (SUTHERLAND; DAVIDMANN; FLAKE, 2006). A linguagem incorpora novos recursos como programação orientada a objetos, uso de threads dinâmicas e comunicação entre processos, e também recursos avançados para a verificação formal de hardware. Os

³ Todos os links de páginas foram testados em dezembro de 2016.

blocos do projeto são especificados por módulos, com portas de entrada e saída associados. Seu funcionamento é definido por um projeto estrutural, referenciando interconexões entre as instâncias dos módulos.

2.5.2 Bluespec SystemVerilog

O Bluespec é uma das linguagens que tem como propósito de acelerar o ciclo de desenvolvimento de projetos de hardware com abstrações em alto nível (DAVE, 2005). As soluções que utilizam linguagens de programação de alto nível de software, como C ou C++, oferecem aos projetistas um alto nível de abstração devido à previsibilidade da arquitetura. No entanto, isso restringe o controle do projetista fazendo com que a integração com a arquitetura seja uma tarefa da ferramenta de síntese (BLUESPEC, 2017). A Figura 8 mostra a arquitetura geral do Bluespec.

Figura 8 – HLS Bluespec

Fonte: Bluespec (2017).

Dentre as vantagens da linguagem Bluespec destacam-se:

- Módulo hierárquico familiarizado com Verilog, controle preciso e transparente sobre a arquitetura.
- Transações Atômicas (regras) em comparação ao Verilog always Blocks e SystemC sc threads. Gera automaticamente controle lógico para o acesso concorrente a recursos compartilhados.
- Métodos atômicos em vez de port lists (Verilog) e métodos (SystemC), reduzindo erros relacionados a interface ou problemas de integração e promovendo a reutilização.

- O Código gerado é totalmente sintetizável em todos os níveis de abstração.
- Simulação antecipada e de grande velocidade através da ferramenta de simulação o Bluesim.

Também oferece um ambiente para projetar, implementar e verificar a arquitetura com alto nível de abstração e síntese em RTL, conforme ilustrado na [Figura 9](#).

Figura 9 – Processo de compilação Bluespec

Fonte: [Bluespec \(2017\)](#).

2.6 Processamento de Imagem

O Processamento de Imagem pode ser definido por uma série de operações matemáticas dependentes de imagens para se obter um resultado desejado. Pode constituir: um melhoramento visual da imagem, a descoberta de características que permitam agrupar imagens em categorias ou uma descrição da cena.

Apesar do progresso significativo que as ferramentas para Síntese de Alto Nível (HLS) alcançaram nas últimas décadas, devido à alta complexidade e aos requisitos computacionais, os projetos de processamento de imagem normalmente requerem um nível de abstração maior do que é possível através da síntese direta de programas C convertidos para RTL, e as ferramentas HLS atuais ainda carecem dessa capacidade. Pesquisas buscam um método de síntese de muito alto nível que permita protótipos rápidos e verificação dos projetos FPGA. Através de resultados

experimentais é demonstrado que é possível efetivamente reduzir a complexidade de projetos e dar vantagens ao uso de FPGAs em relação a outros dispositivos (BI; LI; YANG, 2016).

A definição apresentada por Haralick e Shapiro (1991) descreve uma imagem como a representação espacial de um objeto, cena ou outro fenômeno. Como exemplo podemos citar a fotografia digital (imagem obtida por meio de um sensor de intensidade luminosa digital podendo ser editada, impressa, ou armazenada), a radiografia (representação bidimensional de densidade formada através da exposição de um objeto a raios-X) e os mapas cartográficos (representação de um espaço físico).

O vídeo digital corresponde a uma sequência de amostras de imagens captadas em tempos distintos. A imagem que varia com tempo (t) pode ser representada por uma função de três variáveis contínuas (x_1, x_2, t). A imagem é formada pela projeção do espaço tridimensional (3D) em um plano no espaço bidimensional (2D, representado por x_1 e x_2). Geralmente as variações temporais em cenas 3D são movimentos de objetos na cena, assim, podemos dizer que a variação temporal das imagens reflete a projeção de objetos 3D em uma imagem plana 2D em função do tempo.

2.6.1 Filtros de Convolução

Os filtros locais geram uma nova matriz da imagem através de funções que utilizam a vizinhança de um pixel (representado pela janela W) através de uma varredura na imagem (Figura 10). Com a operação de filtro, o pixel central (representado por x) terá um novo valor que depende do próprio valor e dos vizinhos. Filtros espaciais podem ser implementados através de máscaras (matrizes) utilizando-se dimensões ímpares. Os principais tipos de filtros são: Passa Baixas, Passa Altas, Direcionais e Passa Banda.

Figura 10 – Varredura utilizada em filtros de imagem

Fonte: Bailey (2011).

Nos FPGAs, podem ser utilizados buffers de linha para otimizar a utilização de filtros locais. Esses buffers fazem o armazenamento de pixels e permitam a execução da função em uma única varredura da imagem. A Figura 11 mostra diferentes configurações de buffers de linha. A figura da esquerda refere-se a janela para imagem com deslocamento paralelo e a figura da direita para imagem armazenadas em série.

Figura 11 – Configurações de buffers de linha

Fonte: Bailey (2011).

2.6.2 Módulos de captura

Os sensores digitais de imagem captam fótons e os convertem em sinais elétricos. Com os sensores atuais (CMOS ou CCD), essa conversão geralmente apresenta ruídos. A Figura 12 mostra uma representação do módulo de captura utilizado para a câmera D5M. Após a captura pelo sensor, o circuito envia a imagem particionada em formato digital (RAW) para registradores, juntamente com sinais de controle de linha (LVAL) e controle de quadro (FVAL), ilustrado na Figura 13. Um circuito (CCD Capture) gera coordenadas sequenciais X e Y para os pixels válidos e também fornece um contador de quadros. Os dados captados e as informações geradas podem ser transmitidas para outros módulos, como um conversor para o formato RGB (RAW2RGB).

Figura 12 – Captação de dados pela câmera D5M

Fonte: Elaborada pelo autor.

O processo de varredura da câmera passa por áreas não utilizáveis, chamadas de Branco Vertical e Branco Horizontal. O sinal de controle de linha somente é válido quando não está sobre o Branco Horizontal, da mesma forma, o sinal de controle do quadro somente é válido quando o não está sobre o Branco Vertical.

Figura 13 – Padrão de recepção de imagem pela câmera D5M

Fonte: Elaborada pelo autor.

ESTADO DA ARTE

Exploração do Espaço de Projetos (DSE) têm se apresentado como uma área de pesquisa bastante ativa. Dentre os diferentes sistemas relacionados a essa área, o coprojeto de Hardware e Software apresenta-se como uma tecnologia chave para a pesquisa de novas formas eficientes de construir Sistemas Embarcados para usos específicos.

Neste sentido, muitos trabalhos têm sido desenvolvidos com o objetivo de auxiliar o projetista na busca de soluções eficientes para a construção de sistemas nas mais diversas áreas de pesquisa. Neste trabalho, o enfoque é direcionado à construção de ADAS com os mais diversos tipos de sistemas relacionados à visão computacional.

Observa-se que, à medida que os circuitos FPGA tornam-se maiores em número de portas lógicas e que proporcionalmente os custos tornam-se menores, a complexidade e gama de possíveis implementações criam uma demanda tanto na academia quanto na indústria, para ferramentas que permitam explorar de forma eficiente as soluções relacionadas aos objetivos dos projetos. As principais objetivos estão relacionados a combinação de novos algoritmos com a reutilização de recursos de hardware ou software e também com as personalizações de parâmetros das funções.

O propósito deste capítulo é mostrar o estado da arte das principais ferramentas relacionadas à combinação entre exploração de espaço de projetos com coprojeto de hardware e software e ADAS, com uso de visão computacional. Para isso, inicialmente, é apresentada a estrutura geral de um algoritmo de detecção de pedestre, de forma que se possa compreender os fatores envolvidos no desenvolvimento desses sistemas.

A exploração espacial de projetos está relacionada a três principais questões: esforço de modelagem, esforço de avaliação e precisão dos resultados

A [Figura 14](#) demonstra as vantagens e desvantagens de exploração das ferramentas de DSE ao classificá-las dependendo do nível de abstração da especificação. Nota-se que em níveis superiores de abstração, o projetista pode investigar uma parte maior do espaço de projetos

em um curto espaço de tempo mas com menor precisão. Quanto maior for a necessidade de precisão, melhor detalhado deve ser o sistema, reduzindo assim a abstração e em contrapartida, a modelagem torna-se dispendiosa e a exploração se torna mais difícil. Apesar das ferramentas DSE baseadas em análise fornecerem uma menor exatidão, elas oferecem melhores capacidades de automação e rapidez de exploração (AMMAR; BAKLOUTI; ABID, 2016).

Figura 14 – Características de exploração de ferramentas DSE

Fonte: Ammar, Baklouti e Abid (2016, Adaptado).

3.1 Ferramentas para Exploração do Espaço de Projetos (DSE)

A área de Processamento de Sinal Intensivo (ISP), é uma consequência do volume crescente de informações no qual nos encontramos. A demanda para processamento maciço de sinais digitais, limita-se à restrições de custo e tempo de mercado (*time to market*). Uma das soluções para isto, é o uso de paralelismo em sistemas de vários núcleos, onde os dados são tratadas por meio de cálculos repetitivos.

De forma simplificada, a Exploração do Espaço de Projetos é responsável pela maximização do desempenho da arquitetura do projeto através de ajustes dos parâmetros e fatores que estão relacionadas as métricas do sistema (consumo, tempo de execução, latência, throughput, entre outros).

O projeto de sistemas embarcados requer o uso de modelos que possam atender a diferentes níveis de abstração, distinguindo-se em dois propósitos: especificação e implementação. O propósito da especificação visa descrever o sistema de forma funcional e abstrata, acelerando o desenvolvimento de projetos complexos através de análises baseadas em simulação. Os detalhes de baixo nível, que são necessários para implementar o sistema, são propósitos da implementação.

Os *frameworks* baseados em modelos oferecem uma forma abstrata para projetar sistemas complexos, de controle e para processamento de sinais. Através de uma especificação executável, esses *frameworks* facilitam particionamento entre hardware e software, verificações e permitem iterações mais rápidas nos projetos (BACON; RABBAH; SHUKLA, 2013).

A integração entre hardware em um hardware/software é uma tarefa complexa, esse problema pode ser abordado estendendo funcionalidade aos usuários para que possam especificar quais as partes de suas aplicações devem ser otimizadas através da execução em hardware. Um compilador pode usar o código fonte para criar automaticamente um projeto de hardware otimizado e em conjunto, o código de “cola” necessário para o aplicativo do usuário poder acessar esse hardware. O trabalho de Pu *et al.* (2016), apresenta um sistema que fornece uma semântica de alto nível para explorar mapeamentos diferentes de aplicativos para um sistema heterogêneo, com a flexibilidade adicional de ser capaz de mapear as várias taxas de transferência.

A seguir uma breve explanação de alguns dos sistemas pesquisados para a construção do sistema ADAS-Vision:

Baselabs: Essa ferramenta permite o desenvolvimento de sistemas ADAS e está dividida em 3 partes principais: 1) BASELABS Connect: software de desenvolvimento para uma prototipagem rápida de sistemas, que fornece a infraestrutura para coleta, registro de dados de sensores e permite a fusão de dados complexos de diversos tipos. Também inclui uma interface gráfica de usuário com diversos componentes. 2) BASELABS Create: Permite desenvolver os algoritmos complexos necessários para os sistemas da ferramenta. A Figura 15 mostra a construção de um modelo através do IDE do Microsoft Visual Studio utilizando o recurso para Linguagem Específica de Domínio. Na parte superior é possível criar um modelo através da interconexão de componentes disponíveis na interface. Automaticamente, o código é gerado (parte inferior da tela). Após a complementação do código e das propriedades dos componentes é possível compilar a aplicação. A execução da aplicação está na Figura 16. 3) BASELABS Code: é um gerador de código fonte C genérico para algoritmos complexos, possibilita a redução do esforço manual de geração de código, está em conformidade com os requisitos automotivos típicos (por exemplo,

memória estática), Teste de sistema com grande eficiência. (SCHUBERT *et al.*, 2012).

Figura 15 – Baselabs IDE

Fonte: Schubert *et al.* (2012).

Intempora RTMaps: Framework para prototipagem de algoritmos com um ambiente modular para testar e avaliar funções baseadas em diferentes conjuntos de sensores e com diferentes configurações. Possui uma biblioteca de componentes preparada para desenvolver diversos sistemas. É extensível permitindo a customização de módulos pelo usuário. O kit de ferramentas modular disponibiliza aplicações diversos módulos para: ADAS, veículos autônomos, robótica, UGVs, UAVs, HMI, *data logging*, entre outros. Seu objetivo é facilitar o desenvolver, os testes, as validações, e permitir comparar com outras aplicações. A Figura 17 mostra um exemplo de modelagem para detecção de faixas de trânsito e veículos em um arquivo de vídeo. A ferramenta ainda possui funções específicas para I.A., Visão Computacional, SLAM, fusão de dados, processamento digital, processamento de sinais, mapas digitais e redes neurais. Além disso, permite a interoperabilidade com diversas ferramentas como DDS, MATLAB, SIMULINK, simuladores, QT, QML, ROS e ferramentas de mapeamento digital. Além disso, permite a exportação do código gerado na linguagem C++ (INTEMPORA, 2017).

Comemso ADTF: Framework para sistemas de processamento de imagem no setor automotivo. Permite a criação de filtros de imagem e a conexão com interfaces de hardware para

Figura 16 – Baselabs Aplicação

Fonte: Schubert *et al.* (2012).

sistemas embarcados. Oferece ferramentas para teste e simulação. Dentre as principais funcionalidades destacam-se: Apuração e sincronização de dados de múltiplas fontes de diferentes sensores (radar, lidar, câmera, etc), reprodução de dados em tempo real, processamento de dados, visualização, suporte para C++ no Microsoft Visual Studio, conceito multi-threading e multi-process. Possui uma ferramenta de validação, visualização e teste de ADAS e recursos automáticos de condução, é flexível, eficiente, extensível e estável. Possui diversos toolbox. (COMEMSO, 2017).

NI LabVIEW: Software base da plataforma de projeto da National Instruments, para o desenvolvimento de sistemas de medição ou controle. Integrando diversas ferramentas e com um ambiente de desenvolvimento voltado à resolução de problemas, produtividade acelerada e inovação contínua. Possui blocos de funções designados por instrumentos virtuais, onde cada subprograma pode ser executado isoladamente. Utiliza modelo de fluxo de dados. As principais vantagens: Software integrado de projeto gráfico de sistemas; Suporta ampla variedade de hardware de medição, E/S e barramentos; Interfaces personalizadas, orientadas a evento, para medição e controle; Conjunto de funções matemáticas, análise e processamento de sinais; Compilador de alto desempenho na execução e otimização de código; E, em desvantagem: pequenas mudanças podem ocasionar profundas reestruturações do programa, quando se insere um novo bloco é necessário voltar a ligar os fios

Figura 17 – Intempora IDE

Fonte: Intempora (2017).

e os símbolos para restabelecer o funcionamento. É comum o uso de muitas variáveis, diminuindo-se assim a velocidade de programação e contrariando o modelo de fluxo de dados. (LABVIEW, 2017).

ImprovCV: Sistema dataflow modular para processamento de visão em software. Permite a rápida interatividade com o usuário para o desenvolvimento de aplicações ADAS com visão. É um sistema de processamento de visão de fluxo de dados baseado em componentes. Cada filtro de processamento de visão é seu próprio componente que pode ser conectado de forma dinâmica usando a GUI baseada em fluxo de dados para se conectar entre qualquer outro filtro. Permite a reutilização de componentes de processamento de visão e prototipação rápida e visualização de diferentes configurações de parâmetros. Construído utilizando OpenCV. Os usuários podem arrastar e soltar componentes no gráfico de fluxo de dados e experimentar os parâmetros do filtro. Feedback imediato pela janela de pré-visualização. Filtros podem operar em qualquer tipo de dados, e não são apenas restritos a operar em imagens. Capaz de executar em tempo real, é altamente portátil, usado em: veículos automotivos autônomos, robôs móveis e sistemas de simulação. (BOEING;

Figura 18 – LabView IDE

Fonte: [LabView \(2017\)](#).

[BRAUNL, 2008](#)).

MATLAB HDL Coder e HDL Verifier: Ambiente integrado de projetos que permite acelerar o desenvolvimento para FPGAs e projetos ASIC, integrando ferramentas de projeto e IPs para FPGA da Xilinx e Altera. O HDL Workflow Advisor no HDL Coder converte o código MATLAB de ponto flutuante para ponto fixo e gera códigos VHDL e Verilog sintetizáveis. Dessa forma, permite que a modelagem de algoritmos em alto nível usando construções abstratas e objetos, fornecendo opções para gerar código HDL otimizado para implementação de hardware. O codificador HDL fornece uma biblioteca de elementos lógicos prontos, por exemplo, contadores ou temporizadores. HDL Coder gera automaticamente dois tipos de modelos de co-simulação: 1) Modelo HDL, para realizar a cosimulação HDL com Simulink e um simulador HDL. 2) Modelo FPGA-in-the-loop (FIL), permite verificar o projeto com o Simulink e um com FPGA. ([MATLAB, 2017](#)).

Polis: Sistema de coprojeto hardware e software baseado em máquina de estados finitos. Permite a simulação e síntese de hardware e software e também a especificação em diferentes tipos de linguagens de alto nível como ESTEREL. As máquinas de estado finito podem ser construídas em forma gráfica. Permite subconjuntos para Verilog ou VHDL. Por fim, o Polis é um framework com características de coprojeto, mas sua interface visual permite apenas a criação em máquinas de estado finito (FSM).

Metropolis: O *framework* Metropolis fornece uma infra-estrutura baseada em um modelo com semântica precisa. Suporta a análise de funcionalidades, a descrição de uma arquitetura e

Figura 19 – ImprovCV IDE

Fonte: Boeing e Braunl (2008).

o mapeamento para elementos da arquitetura. Este sistema utiliza uma linguagem lógica capturando restrições não funcionais e declarativas. O foco do projeto concentra-se nas interações entre diferentes formas de trabalhar com os níveis de abstração. Com base em um metamodelo com semântica formal que os desenvolvedores podem usar para capturar projetos, a Metropolis fornece um ambiente para o complexo sistema eletrônico que suporta simulação, análise formal e síntese (BALARIN *et al.*, 2003).

MFSO MDEM: Este trabalho de doutorado, apresenta uma metodologia para projetos especificados por modelagem (MDE) e com suporte a DSE para Sistemas Embarcados. O desafio proposto foi alcançar o balanço entre a flexibilidade da metodologia e o desempenho. Apesar da flexibilidade dos métodos de otimização, esses apresentam baixo desempenho. O acréscimo de funções implica no aumento da complexidade do projeto necessitando um maior gerenciamento. O principais requisitos que geram a motivação de uma ferramenta eficiente estão relacionados à dissipação de potência, desempenho e custos e a pressão sobre o prazo para introdução de um produto no mercado. Diversos estudos de casos são demonstrados com o uso da metodologia, incluindo sistemas que utilizam visão computacional. Nesses estudo, demonstra-se as múltiplas decisões de projeto envolvidas no mapeamento como um único problema de DSE, mostrando adequação para a implementação em ferramentas automática de DSE (OLIVEIRA, 2013).

Figura 20 – MatLab - IDE para HDL

Fonte: MatLab (2017).

PISA: (plataforma e linguagem de programação independente para algoritmos de pesquisa) DSE para exploração espacial de sistemas embarcados e com uma estrutura genérica baseada em tomada de decisão multiobjetivo, otimização de caixa preta e estratégias de pesquisa randomizadas. Especifica uma interface independente do problema entre as estratégias de pesquisa e seleção, Operadores de estimativa e variação de domínios específicos (BLEULER *et al.*, 2003).

Mescal: Metodologia formal e um conjunto de ferramentas de suporte, para o desenvolvimento de plataformas programáveis específicas de aplicativos. O projeto permite a exploração de uma ampla gama de arquiteturas. (Figura 22) (MIHAL *et al.*, 2002).

SPADE: Metodologia para a exploração de arquiteturas de processamento de sinais ao nível

Figura 21 – MatLab - Exemplo de Aplicação

Fonte: [MatLab \(2017\)](#).

Figura 22 – Mescal

Fonte: [Mihal et al. \(2002\)](#).

do sistema. Fornece um meio para construir rapidamente modelos de arquiteturas em um nível abstrato, para mapear facilmente aplicativos, modelados por *Kahn Process Networks*, para esses modelos de arquitetura e para analisar o desempenho do sistema resultante por simulação. A metodologia usa técnica de simulação de ciclos de co-simulação de modelagens de aplicativos e modelos de arquitetura. ([LIEVERSE et al., 2001](#)).

ARTEMIS: O ambiente de modelagem e simulação, com objetivo de explorar eficientemente o espaço de design de arquiteturas de sistemas embarcados heterogêneos em vários níveis de

abstração e para uma ampla gama de aplicações visando essas arquiteturas. (PIMENTEL *et al.*, 2001) .

Daedalus: Ferramenta de projeto de nível de sistema para sistemas multimídia integrados baseados em sistema multiprocessador (MP-SoC). Oferece ferramentas integradas para DSE, síntese no nível do sistema, mapeamento de aplicativos e prototipagem de sistemas de MP-SoCs altamente automatizados. Implantado em sistema de compressão de imagens para câmeras de alta resolução visando aparelhos médicos. Explora o paralelismo de tarefas e dados. Aplicação mapeada para uma variedade de arquiteturas MP-SoC diferentes. Aceleração de desempenho de até 20x em comparação com um sistema de processador único. Os modelos de PM-SoC de alto nível de Daedalus predizem com precisão o desempenho geral do sistema, com erro de desempenho de aprox 5% (NIKOLOV *et al.*, 2008).

SCE: Projeto de nível de sistema baseado em C aborda o desafio da complexidade aumentando o nível de abstração e integrando os processos de projeto para os componentes do sistema heterogêneos. Estrutura de design abrangente. Ambiente baseado na linguagem e metodologia SpecC. Implementa um fluxo de design de sistema de cima para baixo com base em um paradigma de especificação e refinamento Suporte para plataformas heterogêneas, consistindo em componentes de hardware personalizados, processadores de software incorporado, blocos IP dedicados e arquiteturas complexas de barramento de comunicação. A partir de uma especificação abstrata do sistema desejado, os modelos em vários níveis de abstração são gerados automaticamente por meio de um aperfeiçoamento passo a passo sucessivo, resultando em uma implementação de sistema com precisão. A integração das ferramentas automáticas de geração, Estimativa e verificação do modelo, Permite a exploração rápida do espaço de design, Implementação eficiente do MPSoC, Com grande conjunto de exemplos industriais, Ampla gama de arquiteturas de destino, Demonstram ganhos significativos de produtividade no tempo de projeto (DÖMER *et al.*, 2008).

SystemCoDesigner: Usa uma abordagem orientada a atores para integrar a síntese comportamental nas ferramentas de exploração de espaço de projetos. Primeira ferramenta que reduz o número de etapas manuais, Fornece uma geração correta por construção de implementações SoC de hardware/software a partir de um modelo comportamental. Inicia o processo de design a partir de um comportamento do sistema SystemC executável. (KEINERT *et al.*, 2009).

PeaCE: Ambiente de códigos de código HW/SW completo, fornece fluxos de códigos transparentes, simulação funcional, síntese de sistema. Segmentação para aplicativos multimídia com restrições em tempo real, Especifica o comportamento do sistema com uma composição heterogênea de três modelos de computação. Utiliza recursos dos modelos formais

durante todo o processo de design. Ferramentas de design de terceiros podem ser integradas para criar uma cadeia de ferramentas personalizada. Experimentos com exemplos de força da indústria demonstram a viabilidade (HA *et al.*, 2007) .

Koski: Fluxo de design completo para sistemas multiprocessadores (SoCs), Abrange, desde a modelagem do sistema até a prototipagem FPGA, Projeto de sistemas heterogêneos complexos, Aumenta o nível de abstração, Fornece ferramentas de automação de design de nível de sistema, Sistema modelado em ambiente de design UML, especifica as práticas para modelagem de aplicativos e arquitetura ortogonal, As ferramentas de fluxo de design regidas em uma única estrutura que combina as ferramentas em um fluxo contínuo e visualiza o processo de design, Exploração automatizada de arquitetura com base nos modelos de sistema da UML, bem como a anotação automática e posterior de informações no fluxo de design, Exploração da arquitetura baseada na otimização global, Mapeamento de tarefas e agendamento, Implementa o sistema para placa FPGA. (KANGAS *et al.*, 2006).

CASSE: Ambiente de simulação de nível de sistema baseado em SystemC, Ajuda na modelagem e análise de SoCs complexos. Combina modelagem de aplicativos, modelagem de arquitetura, mapeamento e análise em um ambiente unificado, com o objetivo de facilitar e acelerar essas etapas de modelagem. Permitir rápida modelagem e análise no início do processo de design, ajudando na fase de exploração do espaço de design. Implementa em plataforma FPGA. (REYES *et al.*, 2006) .

Essa relação apresenta exemplos de frameworks para processamento de imagens para ADAS, contudo, alguns não estão ligados diretamente à construção de hardware. O objetivo da pesquisa é explorar as principais vantagens desses sistemas, mas também permitindo o coprojetado hardware e software. A outra parte desses sistemas, como o MATLAB HDL Coder e NI LabVIEW tem todas as características para o projeto de pesquisa, contudo, essas ferramentas são comerciais e a construção completa da solução depende da integração de diversos subsistemas.

O projeto ADAS-Vision tem como objetivo fazer DSE para coprojetado, assim, foi realizada uma pesquisa com diversos sistemas para a exploração de projetos abrangendo software e hardware. Além disso, na elaboração do quadro, foi adicionado colunas específicas para identificar os sistemas que já estão em uso para ADAS, para visão computacional e também, sistemas onde o código fonte foi encontrado online. O quadro da Figura 23 mostra o estudo comparativo entre diversas ferramentas para Exploração de Projetos. Dentre as diversas características apresentadas, destacam-se os sistemas que já estão relacionados aos ADAS e também os que trabalham com visão computacional.

Figura 23 – Características de exploração de ferramentas DSE

Ferramenta	Ferramenta de exploração	Hardware e Software	Varios FPGAs	SoPC	IDE gráfico	Geração de Back-end	Processamento Imagem	Projetos ADAS	Código Fonte
Kalray's framework	Algoritmos de agendamento	✗	✗	✗	✗	✗	✗	✗	✗
SDF 3	Algoritmos de agendamento	✗	✗	✗	✗	✗	✗	✗	✗
CASSE	Simulação SystemC	✗	✗	✗	✗	✓	✗	✗	✗
Metropolis	Verificação formal e analisador de rastreamento	✗	✗	✗	✗	✓	✗	✗	✗
Ptolemy	Algoritmos de agendamento	✗	✗	✗	✗	✗	✗	✗	✗
ImprovCV	Simulação C++, profiling	✗	✗	✗	✓	✗	✓	✓	✓
GDSE	Simulações estáticas e dinâmicas	✗	✗	✗	✓	✓	✗	✗	✗
PREESM	Algoritmos de agendamento	✗	✗	✗	✓	✓	✗	✗	✓
GASPARD	Simulação RTL or TLM	✗	✗	✗	✓	✓	✗	✗	✓
SPRINT	Simulação TLM	✗	✗	✗	✓	✓	✗	✗	✓
HELP	Simulação TLM	✗	✗	✗	✓	✓	✗	✗	✓
FAMOUS	Simulação RTL	✗	✗	✗	✓	✓	✗	✗	✓
Baselabs	Simulação C++, profiling	✗	✗	✗	✓	✓	✓	✓	✗
Intempora RTMaps	Simulação C++, profiling	✗	✗	✗	✓	✓	✓	✓	✗
Comemso ADTF	Simulação C++, profiling	✗	✗	✗	✓	✓	✓	✓	✗
MAMPS	SDF 3	✓	✗	✗	✗	✓	✗	✗	✗
MFSO MDEM	Algoritmo Genético	✓	✗	✗	✗	✓	✗	✓	✗
SPADE	Cosimulação baseada em rastreamento	✓	✗	✗	✓	✗	✗	✗	✗
ARTEMIS	Cosimulação baseada em rastreamento	✓	✗	✗	✓	✗	✗	✗	✗
Peace	Simulação de conjunto de instruções, cosimulação precisa de tempo	✓	✗	✗	✓	✓	✗	✗	✗
Mescal	Ambiente com liberdade de exploração	✓	✗	✗	✓	✓	✗	✗	✗
Desert	OBDDs	✓	✗	✗	✓	✓	✗	✗	✗
Explora	Algoritmo Evolutivo em ótimo-de-Pareto	✓	✗	✗	✓	✓	✗	✗	✗
SCE	Simulação de conj. de instruções, verificação formal e profiling	✓	✗	✗	✓	✓	✗	✗	✓
Polis	Simulação Ptolemy, HLS ESTEREL	✓	✗	✗	✓	✓	✗	✗	✓
SystemCoDesigner	Simulação SystemC e otimização multi-objetivo	✓	✓	✓	✓	✓	✓	✓	✓
Koski 1	Simulações estáticas e dinâmicas	✓	✓	✓	✓	✓	✗	✗	✓
MADES	Verificação formal temporal e simulação	✓	✓	✓	✓	✓	✗	✓	✓
Daedalus	Cosimulação baseada em rastreamento e otimização multi-objetivo	✓	✓	✓	✓	✓	✓	✗	✓
NI LabView	Cosimulação baseada em rastreamento	✓	✓	✓	✓	✓	✓	✓	✗
MatLab HDL	Cosimulação baseada em rastreamento	✓	✓	✓	✓	✓	✓	✓	✗
ADAS-Vision	Simulação RTL, SystemVerilog, árvore decisão	✓	✓	✓	✓	✓	✓	✓	✓

Fonte: Ammar, Baklouti e Abid (2016, Modificado).

IMPLEMENTAÇÃO DO PROJETO

Neste capítulo, descreve-se a construção e o processo de funcionamento do *Framework ADAS-Vision*. A [Figura 24](#) visa servir de referência para facilitar o entendimento das explicações. Esta figura representa todos os sistemas associados ao projeto.

Figura 24 – ADAS-Vision Framework

Fonte: Elaborada pelo autor.

O IDE *framework*, indicado pela legenda como **ADAS-Vision**, realiza a integração entre IP Software e IP Hardware em tempo de execução. Para isto, o **Simulador de Coprojeto** executa os IPs Software e aciona externamente o **Simulador de Hardware**. Este simulador é compatível com o código Verilog gerado pelo **Bluespec** para a construção de **IPs** de Hardware. Após a simulação, o usuário da ferramenta aciona o **Gerador de projetos**, que constrói automaticamente os projetos que serão executados na **Placa de desenvolvimento FPGA**. Para isto, dependendo do projeto implementado pelo usuário, podem ser gerados três tipos de projeto: 1) Para execução no processador **Intel Atom**, é criado um **Projeto Software** que deve passar por um **Compilador**

C++; 2) Quando o projeto contém IPs Hardware, é gerado um **Projeto Hardware** que deverá passar pela síntese na ferramenta **Quartus II**; 3) Se o projeto implementa um software para processador embarcado, cria-se um **Projeto SoPC** que deverá ser compilado pela ferramenta **NIO II** e posteriormente carregada no FPGA. O *framework* também possui um banco de modelos de projeto em uma base de **Gabaritos**. Ao selecionar um gabarito (**Template**) o usuário pode fazer suas personalizações.

Com base nessa introdução, este capítulo foi organizado da seguinte forma: A [seção 4.1](#) apresenta a **Placa de desenvolvimento FPGA**, que é o ambiente de execução dos projetos. Na [seção 4.2](#) encontra-se a documentação da construção do software **IDE ADAS-Vision**. A [seção 4.3](#) detalha o funcionamento do **Simulador de Coprojeto**. E ao final, na [seção 4.4](#) descreve-se as diversas formas que o **Gerador de projetos** pode disponibilizar o código final.

4.1 Plataforma de Hardware

Os experimentos conduzidos neste trabalho, utilizaram a placas de desenvolvimento Terasic DE2i-150, que contém um FPGA Altera Cyclone IV (modelo 4CX150) e um processador Intel Atom (modelo N2600) integrados. A intercomunicação entre estes dispositivos é realizada através de um barramento PCI Express (Geração 1). Os recursos associados ao processador e ao FPGA são isolados, dessa forma, há controles independentes para cada um dos circuitos. A [Figura 25](#) mostra com mais detalhes os blocos disponíveis.

Figura 25 – Componentes da placa de desenvolvimento DE2i-150

Fonte: [Terasic \(2016\)](#).

O FPGA Cyclone IV (modelo EP4CGX150DF31) dispõe de 149.760 elementos lógicos,

720 blocos de memória (M9K), 6.480 Kbits de memória embarcada, 8 PLLs (Phase Locked Loops). Interconectado a este, a placa também tem 128MB de SDRAM (32M x 32 bit), 4MB SSRAM, 64MB de memória Flash (16 bit) e três osciladores de clock de 50MHz.

O processador Atom N2600 dispõe de dois núcleos de processamento de 64 bits funcionando a 1.6GHz. Executando 4 threads simultâneas através da tecnologia Hyper-Threading. Contém ainda, 1MB de memória cache, um processador gráfico integrado executando à 400MHz e extensões do conjunto de instruções SSE2, SSE3 e SSSE3. Interconectado ao processador, a placa oferece 2GB de memória SDRAM DDR3, conexão wireless e outros recursos.

Dentre os demais recursos, foi utilizado um Kit com display LCD colorido (modelo TRDB-LTM) e uma câmera CMOS de 5M pixel (modelo TRDB-D5M) ambos da Terasic.

4.2 Construção do Software

Um dos objetivos principais da ferramenta é a simulação de coprojeto. Para atender a este requisito, o *framework* dispõe de bibliotecas de IPs. Existem 3 tipos diferentes disponíveis no sistema: IPs Software, que implementam rotinas desenvolvidas em C++, mas quais podem fazer uso de bibliotecas externas, como por exemplo o openCV; IPs Hardware, geralmente implementados em Verilog através do Bluespec, onde são enviados para o simulador de hardware; E os IPs de Entrada e Saída que, são especiais por fazerem as comunicações do ADAS-Vision com sistema de arquivos e com dispositivos USB (Figura 26).

As diferenças internas entre os tipos de IPs são transparentes para o usuário da aplicação. Todos os tipos contém o mesmo tipo de interface, tornando a ferramenta de fácil aprendizado para quem não é especialista na área (*learnability*). Da mesma forma, não há diferenças para as interconexões de tipos diferentes de IP, possibilitando uma das funcionalidades chave do sistema, que é a capacidade de validar combinações híbridas antes de se preocupar com otimizações.

Cada IP funciona de forma independente, ou seja, a ferramenta permite facilmente expandir as bibliotecas sem afetar as já existentes. Também é possível utilizar recursos externos através da linguagem de programação, como foi o caso da utilização de importação e exportação de arquivos do formato MatLab. Espera-se em um futuro próximo que outras ferramentas como o Project-R, Weka ou Octave, possam fazer parte das opções de expansão (capacidade de expansão).

A ferramenta de implementação deste trabalho também permite facilmente a portabilidade para outras plataformas como Linux, Web ou dispositivos móveis. Esse foi um dos motivos para a seleção da ferramenta QT de programação (*interoperabilidade*) (QT, 2017). Ainda como opção para outras plataformas de desenvolvimento, a ferramenta também pode ser compilada em diferentes compiladores, como o Microsoft Visual Studio ou GNU-Compiler.

Figura 26 – Tipos de bibliotecas de componentes

Fonte: Elaborada pelo autor.

Exploração do Espaço de Projetos

O pré-requisito para utilização de um projeto no ADAS-Vision é a escolha do dispositivo FPGA. Este passo é necessário para que o sistema possa indicar os limites estimados do projeto. Apesar das estatísticas armazenadas individualmente de cada IP, o projeto final passa por diversas fases durante a síntese que pode ocasionar uma alteração dos recursos necessário para executar no FPGA. Contudo, esta seleção de dispositivo serve para balizar o usuário, para que se tenha uma estimativa, mesmo que aproximada, da viabilidade de executar o projeto em um sistema embarcado. A [Figura 27](#) mostra o processo de DSE na ferramenta.

A opção de carregar um gabarito já pronto, serve para facilitar a elaboração de variantes de um projeto. Inicia-se com as configurações originais e é possível criar diversas versões de um mesmo sistema. Dessa forma, é possível comparar as vantagens e desvantagens de cada abordagem.

O método de exploração da ferramenta é o processo iterativo. Com o projeto carregado, o usuário simula o sistema e valida a saída. Caso a solução não seja satisfatória, o usuário tem

Figura 27 – Processo de DSE na ferramenta

Fonte: Elaborada pelo autor.

a opção de fazer novas manutenções no projeto e em seguida simular novamente. Esse ciclo termina quando o usuário estiver satisfeito com a resposta obtida na simulação. A partir deste ponto, o usuário pode proceder com a compilação para testes na plataforma de hardware.

Diversos sistemas apresentam formas automáticas de DSE, conforme detalhado no [Capítulo 3](#). Futuramente, espera-se que tais formas possam ser agregadas à ferramenta. Como por exemplo, o uso de computação evolutiva, aprendizado de máquina ou ferramentas de análise estatística.

Processo de utilização da ferramenta

Em linhas gerais, a elaboração de um projeto dentro do *framework*, segue um fluxo comum. Para concluir todo o processo (elaboração + testes + execução) de um coprojeto, faz-se necessário o uso de ferramentas externas. A [Figura 28](#) detalha passo-a-passo o fluxo de utilização, dentro e fora da ferramenta:

No primeiro passo, após iniciar a ferramenta é necessário selecionar a placa FPGA a

Figura 28 – Fluxo geral de funcionamento do sistema proposto

Fonte: Elaborada pelo autor.

qual se destina o projeto. Com a definição das especificações do projeto é possível selecionar um gabarito (modelo) com um sistema pré-montado onde é possível posteriormente fazer modificações. Os gabaritos são projetos já validados que contêm exemplos de utilização da ferramenta para diversos sistemas ADAS baseados em visão.

O sistema disponibiliza uma biblioteca de IPs onde é possível adicioná-los ao espaço de trabalho e interligá-los a outros blocos. O IP pode ser hardware ou software, dependendo da escolha do usuário e da disponibilidade na biblioteca. Cada IP possui um conjunto de parâmetros que podem ser modificados durante a manutenção do projeto.

No segundo passo, depois de concluída a esquematização, será possível fazer uma simulação do projeto modelado. O sistema irá usar as rotinas pré-compiladas de IPs na ferramenta BlueSim para hardware e também os programas compilados para o software para fazer a simulação completa do projeto modelado. Nesta parte do projeto, será possível selecionar uma base de imagens sequenciais onde será possível visualizar o resultado de cada parte do projeto modelado.

O terceiro passo é a geração do projeto para ser executado em dispositivos reconfiguráveis. Nesta etapa são preparados os arquivos para serem usados na ferramenta de síntese de Alto-Nível (HLS) e também os scripts de compilação do código C para processadores soft-core.

O quarto passo é a geração de código Verilog dentro da ferramenta Bluespec. A proposta é que o usuário precise apenas chamar a ferramenta e, sem a necessidade de fazer modificações, compile o projeto. Contudo, é possível fazer uso das diversas funcionalidades da ferramenta como simulador e estimador de consumo de hardware.

Por final, o quinto passo compreende utilizar o código Verilog gerado e sintetizá-lo no Quartus II da Altera para ser posteriormente transferido para o FPGA. Após esta etapa também é necessário executar o script gerado no passo 3 para fazer a inserção do software no processador soft-core já embarcado.

4.2.1 Especificação da Interface Gráfica

A construção do ambiente de interação com o usuário foi elaborado com a ferramenta QT utilizando a linguagem C++ (QT, 2017). Em resumo são partes da interface:

Espaço de Trabalho: Local da área em que o projetista poderá lançar os blocos IP disponíveis e determinar a ligação entre eles. Cada IP tem uma especificação de sua interface de comunicação, na tentativa de interligação de IPs o sistema faz a validação da compatibilidade entre as conexões. Nesta área também será possível organizar livremente os blocos de IP conforme escolha do usuário. A Figura 29 mostra um esboço dessa interface.

Figura 29 – Esboço da Interface da aplicação

Fonte: Elaborada pelo autor.

Menu Principal: As bibliotecas de IPs disponíveis estão no menu principal onde há uma lista de blocos em hardware ou software pré-construídos. Nesta lista os IPs poderão ser adicionados ao Workspace criando novas instâncias de objetos. O catálogo dos IPs pode ser

livremente expandido conforme novas versões da ferramenta. A interface gráfica dos IPs atende a um padrão pré-estabelecido onde as entradas e saídas disponíveis, ficam nas laterais. O conteúdo gráfico do centro do objeto de IP depende das particularidade implementadas em cada tipo. A Figura 30 mostra um esboço do bloco de um IP na ferramenta, onde é possível observar duas entradas (*Image* e *Value*) e uma única saída (*Image*). Na parte central há uma pré-visualização em miniatura do resultado de processamento do bloco. Na parte superior está a identificação do bloco, sendo que cada tipo possui uma cor diferente para facilitar a identificação pelo usuário.

Figura 30 – Esboço de um IP no sistema proposto

Fonte: Elaborada pelo autor.

Componente de IP: A manutenção das propriedades de um IP pode ser realizada ao selecionar a instância deste componente no Espaço de Trabalho. Neste componente, a manutenção de diversos atributos/parâmetros pode ser realizada através de objetos interativos. Como por exemplo, caixas de texto, botões, controles deslizantes, etc.

Visualização: A aplicação fornece um IP especial para visualização da imagem que está sendo processada: *SW Image View*. Quando o projeto estiver no modo de simulação, esta janela permitirá visualizar graficamente o comportamento da imagem durante o processo. Este bloco pode ser conectado em qualquer outro onde exista saída de imagem. Dessa forma, pode-se visualizar a imagem em diversas etapas do processo, indo desde a sua forma de entrada, até finalmente a imagem completamente processada pela aplicação modelada.

Barra de menus: Na parte superior da tela estão localizados os menus para manutenção do projeto. O menu *File -> New Project* (Novo projeto) inicia a construção de uma nova modelagem pelo projetista ou o carregamento de um projeto já gravado. Ao clicar nesse botão haverá uma tela para seleção de gabaritos pré-construídos ou projetos gravados. Após selecionar um gabarito, este será instanciado no *Workspace* permitindo sua modificação pelo usuário. O menu *Build* (Construir) inicia a geração do projeto para a síntese/compilação do projeto modelado pelo usuário. Esse menu cria os arquivos de projeto em pastas pré-definidas para

posterior execução em na placa FPGA. Por fim, no menu principal há submenus para salvar o projeto ou a criação de um novo *template* (gabarito) a partir do projeto em uso, permitindo sua reutilização posterior.

A Figura 31 mostra um exemplo completo da interface, com um exemplo de projeto criado pelo o usuário.

Figura 31 – Esboço de um projeto esquemático no sistema proposto

Fonte: Elaborada pelo autor.

4.2.2 Especificação do Projeto de Software

A ferramenta de projetos ADAS-Vision foi implementada em C++ usando diversas bibliotecas, entre elas o OpenCV. Na Figura 32 há uma representação simplificada das classes do projeto. Como cada IP gera uma classe diferente, apenas alguns IPs foram representados para exemplificar.

As classes dos componentes de IPs de software são identificadas pelas iniciais **SW**, as de hardware **HW**, e por fim, as classes que fazem Entrada/Saída de dados como leitura de arquivos de vídeo ou comunicação com outros dispositivos como o PCIe são identificadas pelas iniciais **IO**.

As classes de IPs herdam a classe *GraphicsNode* que contém a programação principal de cada IP. Essas classes são instanciadas em um objeto do tipo *QGraphicsScene*. A interligação dos IPs (aqui denominados *nodes*) é feita pela classe *GraphicsBelzierEdge*.

Com o fim do processamento, a imagem é armazenada em uma variável interna através da propriedade [setImageOut]. Ao fazer isto, o componente de junção entre IPs (*GraphicsDirectedEdge*) propaga a alteração para o próximo IP. A instância do IP seguinte (se houver), é chamada pelo do método [onSourceDataChanged] (que está no componente referente à linha de conexão entre IPs: *GraphicsDirectedEdge*). Este método faz a leitura da imagem alterada e atribui o valor ao método [setImageIn] do próximo componente. Iniciando-se assim a mesma sequência para todos os demais componente interconectados.

Para generalização, a classe *GraphicsDirectedEdge* faz a chamada através da leitura de metadados dos sockets (*GraphicNodeSocket*), onde é instanciado pelo construtor do IP.

Ao final do processamento e da propagação, a imagem recém processada é convertida para o padrão de visualização da aplicação. Isto é necessário pois a aplicação não implementa a classe de visualização direta do openCV. Essa característica certamente pode causar uma sobrecarga de processamento, mas garante o desacoplamento e a independência dos componentes externos.

Classe do IP de Hardware

A grande mudança entre a classe de IP Software e de IP Hardware está dentro da rotina [imageProcessing]. Basicamente, todos os outros procedimentos são iguais em sua forma de execução. A [Figura 34](#) mostra com detalhes a propagação de uma imagem pelo componente de IP Hardware.

Dentro da subrotina [imageProcessing], é executado uma chamada para uma sub-rotina de exportação da imagem do formato OpenCV para um arquivo binário. Isto se faz necessário pois o processador virtual do IP Hardware está preparado para ler nesse formato. Feito isso, através de uma API com o sistema operacional, é criado um comando *Shell* para chamar o executável do compilador, através de um script genérico de execução.

Para a execução de IP Hardware, normalmente existem dois scripts: Um para a recompilação do IP hardware (caso algum parâmetro mude e seja necessário a recompilação); E um para a execução do processador virtual. Esses scripts foram gerados durante a construção a concepção do projeto do IP para a ferramenta o *framework*.

Uma vez que a recompilação do Hardware é concluída, chama-se através de API e script, a execução do processador virtual verilog (.vvp). Para ambos, a API é parametrizada para criar um bloqueio na *thread*, impedindo que o sistema continue enquanto houver processamento externo. Uma vez liberada a *thread*, inicia-se a rotina que faz a leitura do arquivo processado, para posterior conversão de volta ao formato do OpenCV. Assim como no IP Software, a imagem do formato OpenCV é convertida no padrão da interface da ferramenta.

O funcionamento detalhado será ilustrado na [seção 4.3](#), onde é esboçado graficamente o funcionamento de IP Software e de IP Hardware. Assim como as opções de recompilação de hardware em tempo de execução da ferramenta.

Figura 34 – Diagrama de Sequencia de um IP Hardware

Fonte: Elaborada pelo autor.

4.3 Simulador de coprojetos

O simulador de coprojetos faz a execução do projeto (modelado) dentro da própria ferramenta. Sem que seja necessário qualquer chamada externa por parte do usuário. A simulação em software é feita pelos próprios componentes IP Software, sem necessidade do uso de ferramentas externas. Já para a execução do hardware em HDL, o ADAS-Vision utiliza como ferramenta o simulador ICARUS Verilog (ICARUS, 2017).

O Icarus Verilog (*iVerilog*) é uma ferramenta de simulação e síntese de Verilog. As principais características levadas em consideração para sua utilização no *framework* são: Disponibilidade do código fonte; Disponibilidade para outras plataformas; Compatibilidade com o Bluespec; Maturidade de código (atualizações desde 1990); Execução através de linha de comando (não necessário interação manual).

O princípio de funcionamento do iVerilog resume-se a compilação do código escrito no padrão Verilog (IEEE-1364) em um formato intermediário, que possa ser executado pelo simulador VVP (*Virtual Verilog Processor*). O código HDL compilado pode ser através de uma única linha de comando. O formato original da maioria dos IPs Hardware são escritos na linguagem Bluespec e copilados na ferramenta para o Verilog. No ambiente da ferramenta Bluespec é possível fazer a simulação com o iVerilog.

Para que fosse possível o intercâmbio de dados entre os IP Software e o VVP, o núcleo do código Verilog (do IP Hardware) é encapsulado em um controlador de bancada (também escrito em Verilog) ([Figura 35](#)). Este controlador, instância o núcleo, transmitindo dados digitais que faz a captura através da leitura e escrita no sistema de arquivos. Assim, quando o componente IP Hardware (no ADAS-Vision) recebe uma imagem ou algum tipo dado, este componente faz a conversão para o formato que será lido pelo VVP e em seguida executa-o. Ao fim do processo, o arquivo de resposta deixado pelo simulador é então lido e convertido para o formato OpenCV.

Figura 35 – Funcionamento do Controlado de Bancada

Fonte: Elaborada pelo autor.

O **Código-fonte 1** refere-se a um controlador de bancada na linguagem Bluespec. Na linha 9, nota-se que há um ponteiro para a leitura do arquivo [in.dat]. As linhas 17,18 e 19 são responsáveis pela leitura informações através do comando *\$fgetc(..)*. A linha 29, faz a gravação dos dados no arquivo através do comando *\$fwrite(..)*. A instância do núcleo do IP é criada pela linha 4, onde o nome do núcleo é *mkCalcular* e a interface que determina o tipo é *Processor*.

Código-fonte 1 – Subrotina de um bloco de controlador de bancada em Hardware

```

1:  import processor::*;
2:  (* synthesize *)
3:  module rgbConvert(Empty);
4:    Processor p <- mkCalcular();
5:    Reg#(Bit#(3)) cnt <- mkReg(0);
6:    Reg#(File) arq <- mkRegU;
7:    Reg#(File) arqOut <- mkRegU;
8:    rule openFiles (cnt < 1);
9:      File f_in <- \fopen("in.dat","r");
10:     arq <= f_in;
11:     File f_out <- \fopen("out.dat","w");
12:     arqOut <= f_out;
13:     \display("Iniciado ...");
14:     cnt <= 1;
15:   endrule
16:   rule executar (cnt == 1);
17:     Int#(32) i <- \fgetc(arq);
18:     Int#(32) j <- \fgetc(arq);
19:     Int#(32) k <- \fgetc(arq);
20:     if (i == -1 || j == -1 || k == -1)
21:       cnt <= 3;
22:     else
23:       begin
24:         p.put(i,j,k);
25:         cnt <= 2;
26:       end
27:   endrule
28:   rule saveData(cnt == 2);
29:     \fwrite(arqOut, "\%s",p.get());
30:     cnt <= 1;
31:   endrule
32:   rule fim (cnt == 3);
33:     \fclose(arq);
34:     \fclose(arqOut);

```

```

35:      \display("fim");
36:      \finish(0);
37:      endrule
38:  endmodule

```

A principal vantagem dessa abordagem é o total desacoplamento, permitindo independência nas alterações tanto de hardware quanto de software. Isso contribui para que os componentes de hardware possam ser modificados, recompilados e simulados durante o tempo de execução da ferramenta de software.

A [Figura 36](#) demonstra o conceito envolvido. Para exemplificar, quando há um estímulo na conexão de entrada, os dados (da imagem) e dos parâmetros do componente, são enviados para o sistema de arquivos através do sequenciador (arquivo in.dat). O serializador apenas converte os dados para um formato serial hexadecimal. O proxy então, aciona o VVP que executa o controlador de bancada, ocasionando a leitura dos dados do sistema de arquivos. O programa do controlador faz a leitura serial do arquivo e envia em formato digital para o núcleo do hardware. Em resposta, o hardware devolve as informações processadas que serão convertidas em arquivo de saída(arquivo out.dat). Quando o proxy percebe que o VVP terminou sua tarefa, ele aciona novamente o sequenciador que, agora, faz a leitura (do arquivo out.dat) e sua desserialização para o objeto de imagem do OpenCV.

4.3.1 Customização de IP em tempo de Execução

No exemplo anterior, considerou-se que os parâmetros do IP são apenas informações de entrada. Mas, quando um parâmetro altera a programação do verilog, como por exemplo o tamanho de uma FIFO, então o processo tem mais uma fase. Esta fase, é responsável pela recompilação do hardware. O [Código-fonte 2](#) apresenta um exemplo de código para ser usado em conjunto com o Controlador de Bancada previamente apresentado. Este exemplo implementa a conversão de RGB para escala de cinza. Em anexo, o [Código-fonte 3](#) apresenta a versão do mesmo código compilada para a linguagem Verilog.

Código-fonte 2 – Subrotina de um bloco IP em Hardware

```

1:  interface Processor;
2:      method Action put(Int#(32) r, Int#(32) g, Int#(32) b);
3:      method Bit#(8) get();
4:  endinterface
5:  (* synthesize *)
6:  module mkCalcular(Pprocessor);
7:      Reg#(Int#(32)) value <- mkRegU;
8:      method Action put(Int#(32) r, Int#(32) g, Int#(32) b);
9:          value <= (r + g + b) / 3;

```

Figura 36 – Simulação de um IP em Hardware

Fonte: Elaborada pelo autor.

```

10:     endmethod
11:     method Bit#(8) get();
12:         return truncate(pack(value));
13:     endmethod
14: endmodule

```

Uma das grandes vantagens da ferramenta, é a possibilidade de personalizar IP Hardware em tempo de execução (*runtime*). Durante a criação de um fluxo projeto, alguns IPs podem gerar alterações consideráveis no código do núcleo. Este tipo de operação, implica na necessidade de recompilação do código Verilog para que o mesmo possa ser simulado. A ferramenta ADAS-Vision faz isso de forma transparente para o usuário, sendo desnecessário chamar outra ferramenta.

Internamente, a programação do IP de hardware acessa um banco dos arquivos fontes originais em Verilog e faz uma cópia para a área do projeto do usuário, especificamente em uma pasta codificada com a instância do objeto acessado. Dessa forma, é possível alterar a cópia sem alterar os arquivos originais. Quando o usuário faz alterações nos parâmetros do bloco selecionado, o IP altera os arquivos de controlador de bancada e do núcleo do bloco incrementando essas modificações (1), em seguida é chamado o arquivo de lote de compilação deste IP que gera o processador virtual para ser executado posteriormente (2). Quando não há necessidade de recompilação, ou quando o IP não permite modificações, o processo normal do programa de simulação do IP é chamar o sequenciador. Este sequenciador faz a conversão dos dados de imagem (geralmente matriz do OpenCV) para um fluxo de dados que possa ser lido pelo processador virtual (3). O sequenciador aguarda os dados processados pelo bloco de hardware e faz a conversão novamente para o OpenCV para saída de dados do bloco (Figura 37).

Figura 37 – Customização de IP em tempo de execução

Fonte: Elaborada pelo autor.

Os blocos de hardware também podem ser modificados manualmente através na pasta do projeto criado. Nesta pasta há um script executável que faz automaticamente a recompilação.

O [Código-fonte 5](#) (vide Apêndice A) demonstra este script de recompilação do IP Hardware (Verilog), preparando-o para que possa ser usado pelo VVP. E, por final, o [Código-fonte 6](#) mostra o script para a execução do IP no simulador. Nota-se que, ao executar o segundo script (execução), o processador virtual apenas carrega o arquivo [in.dat], processa e libera o arquivo [out.dat]. Todos os outros passos, como por exemplo serializador e desserializador, são feitos através do software.

Co-Simulação com pré-processamento

A Co-Simulação com pré-processamento oferece ao usuário a possibilidade de criar uma simulação no processador Atom (podendo o usuário selecionar blocos de hardware ou software simulados) e em seguida usar a comunicação via PCI-Express para acionar um módulo de hardware instanciado no FPGA. Para isso é necessário que o usuário faça o carregamento prévio de um projeto sintetizado no FPGA e use o IP de conexão PCIe no projeto de simulação.

No projeto de hardware, os cálculos aritméticos são programados em um bloco denominado *Top Level Design*, que é um sub-bloco no Módulo de Usuário (*User Module*). Este módulo é o núcleo do projeto, contendo toda a lógica personalizada do IP. Ele interage com o(s) sub-bloco(s) para fazer operações computacionais.

O Módulo possui também, um bloco *slave* (escravo) para leitura/gravação (através de um byte de controle para início/parada) e registros de informação da imagem. Este módulo é responsável por iniciar operações em memória SDRAM.

Para o Software, o processador faz a leitura e escrita da memória via barramento PCIe através do IP disponível na ferramenta de simulação, criando uma ponte de comunicação com os módulos de hardware instanciados.

Figura 38 – Co-Simulação com pré-processamento

Fonte: Elaborada pelo autor.

Neste exemplo, o simulador carrega dados de imagens (1) e faz o processamento no simulador através de blocos IP (2), em seguida, os dados são enviados pelo IP de comunicação

PCIe (3). Ao fazer isto, os dados são transmitidos através do barramento para o IP em hardware do lado FPGA. O bloco com a lógica personalizada (instanciada previamente) processa os dados transmitidos (5 e 6) e os envia para a saída VGA do FPGA (Figura 38).

Este tipo de simulação permite que o usuário veja os resultado do processamento do hardware embarcado em tempo de execução.

Co-Simulação usando FPGA como dispositivo (*runtime*)

Esta simulação assemelha-se a de pré-processamento, contudo, neste caso, quando os dados pré-processados são enviados para o FPGA (passos 1, 2, 3, 4) a lógica embarcada faz o processamento (5,6) mas não envia para o VGA do lado FPGA, fica aguardando a leitura do simulador através do PCI-Express (7). Após a leitura dos dados processados em tempo de execução (*runtime*) o IP do PCIe continua o fluxo de simulação criado pelo usuário com a imagem processada pelo FPGA (9). Por final, os dados são enviados para o driver de VGA (10) e finalmente exibido ao usuário (11) (Figura 39).

Figura 39 – Co-Simulação usando FPGA como dispositivo (*runtime*)

Fonte: Elaborada pelo autor.

Este tipo de simulação é muito interessante para fazer validações de IPs Hardware que ainda não estejam integrados com o ADAS-Vision, lembrando que neste caso, não é necessário que o hardware que está no FPGA tenha sido gerado pelo *framework*, desde que esteja encapsulado no padrão de comunicação do PCI-express do simulador poderá se comunicar com ele através do barramento.

É importante observar que a placa DE2i-150 tem duas saídas VGA. Uma usada pelo processador Atom e outra pelo FPGA. Neste exemplo foi usada a saída do lado Atom, mas o sistema permite que sejam utilizadas as duas saídas simultaneamente, permitindo comparações das imagens de saída pelo usuário.

Co-Simulação usando barramento AMBA no FPGA

O protocolo AMBA é um padrão industrial aberto, com especificação para interconexão entre circuitos e gerenciamento de blocos funcionais sendo utilizado amplamente em diversos projetos ASIC e SoC. Ele permite o desenvolvimento de projetos multiprocessados com grande número de controladores e periféricos.

Nesta simulação o Atom envia comandos e dados através de PCIE (passos 1,2,3,4) até chegar ao barramento Avalon e, em seguida é feita uma conexão em ponte com o arquivo de registro escravo (*slave*) (5). O bloco lógico (criado pelo usuário) acessa as informações disponíveis no arquivo de registros e faz o processamento (6) registrando em memória (7,8). Finalmente, o Atom envia solicitação de leitura para o arquivo de registro escravos para obter os dados processados (Figura 40).

Figura 40 – Co-Simulação usando barramento Avalon no FPGA

Fonte: Elaborada pelo autor.

Uma das vantagens da utilização desse barramento é a possibilidade de ganho de desempenho em alguns projetos pela sua natureza de trabalhar com *pipelines*. Além disso, a lógica do usuário pode pedir ao barramento mestre para enviar comandos e dados para outros componentes escravos conectados ao barramento como, por exemplo a SDRAM que possui grande capacidade de armazenamento neste dispositivo.

Co-Simulação usando processador embarcado

O processador *softcore* Nios II é um núcleo de processador RISC de uso geral desenvolvido pela Altera. O processador pode ser customizado e sintetizado em um FPGA ou ASIC através de um pacote de aplicativos fornecidos pelo desenvolvedor. Sua flexibilidade possibilita fácil conexão com outros periféricos, alteração do conjunto de instruções e, alteração das estruturas internas, como tamanho da memória cache, priorização de interrupções, dentre outros (ALTERA, 2016a).

A simulação deste tipo de IP contém um código de programação pré-programado equivalente às funções do NIOS II. Neste caso, para funcionamento do simulador o código pode ser levemente diferenciado para comportar as diferenças de plataformas de execução (Figura 41).

Figura 41 – Co-Simulação usando NIOS II

Fonte: Elaborada pelo autor.

Nesta abordagem, por tratar-se de um IP de coprojeto (hardware e software) embarcado, é possível fazer diversas personalizações permitindo explorar o espaço de projetos (DSE).

Um das grandes vantagens do processador NIOS II é a possibilidade de criar instruções customizadas em hardware para acelerar a execução de programas, conforme mostrado na Figura 42. A execução das instruções customizadas se assemelham ao uso de um co-processador, onde uma lógica customizada em hardware pode ser usada para se comunicar com o software embarcado (ALTERA, 2016b).

Figura 42 – Co-Simulação usando NIOS II com instrução customizada

Fonte: Elaborada pelo autor.

Processo de Construção de um IP

A validação de algoritmos de visão computacional é usualmente implementada em arquitetura de propósito geral, mais especificamente em computadores pessoais (desktops ou notebooks). Atualmente, existem diversos pacotes de software como o OpenCV ([OPENCV, 2017](#)) ou MatLab ([MATLAB, 2017](#)) para facilitar a modelagem de projetos com a utilização de algoritmos. A implementação nesse tipo de arquitetura é relativamente mais simples de ser realizada quando comparada com a implementação *on-chip*, pois, neste caso, o projetista não precisa se preocupar com os detalhes intrínsecos do hardware, tais como os problemas relativos ao sincronismo de circuitos ou ao particionamento de hardware e software (coprojetos). No entanto, quando um dos requisitos de projeto é o uso em sistemas embarcados, torna-se necessário levar em consideração a capacidade de processamento, o consumo de energia, peso e o tamanho. As arquiteturas de propósito geral não se mostram adequadas para atender a esses requisitos, contudo, sua principal vantagem é a flexibilidade de manutenções e na adequação a novos recursos.

Durante a construção dos IPs, especificamente na simulação com o Bluesim, é gerado um arquivo executável de simulação para cada bloco de hardware. Nesse ponto, esses arquivos serão automaticamente chamados pelo simulador de coprojetos passando parâmetros de chamadas através de arquivos de configuração pré-estabelecidos. A [Figura 43](#) mostra o fluxo completo da construção de um IP Hardware.

Após a especificação e a codificação de um IP de Hardware este deverá passar por testes no simulador Bluesim. Posteriormente, o mesmo deverá ser sintetizado na ferramenta do Quartus II e testado na placa de FPGA. Todos esses passos são recomendados para minimizar problemas que possam ocorrer na modelagem do usuário dentro do framework, evitando-se ao máximo erros que tenham origem nos Blocos IPs de Hardware.

Inicialmente o IP Hardware pode ser construído no Bluespec IDE, onde passar por um ciclo de testes e validações. O ADAS-Vision permite a utilização de outros códigos Verilog, contudo, se possível, recomenda-se o uso do Bluespec pois o fornecedor garante (com poucas exceções) que o código será sintetizável. Neste ambiente é possível criar o Controlador de Bancada para realizar testes e também para preparar o novo IP para execução no *framework*. Em seguida, o IP pode ser testado no ambiente do Quartus II. As estatísticas de tempo, elementos lógicos, latência, *throughput*, frequência máxima entre outras devem ser registradas para uso como base de dados para o ADAS-Vision estimar se os recursos de um projeto implementado são viáveis. Após todos os testes no FPGA, é necessário fazer a implementação da chamada da rotina no ambiente QT de desenvolvimento C++.

Uma das maiores dificuldades para a implementação completa de um IP na ferramenta, estão associadas as rotinas de geração de código automático, pois cada cenário ou tipo de interconexão pode gerar diferentes implementações.

Figura 43 – Fluxo completo para compilação de um projeto em Hardware

Fonte: Elaborada pelo autor.

A Figura 44 fornece um diagrama mostrando as extensões dos arquivos formados a partir do Bluespec.

Figura 44 – Construção de IPs

Fonte: Elaborada pelo autor.

4.4 Gerador de Projetos

O gerador de projetos (menu *Build*), faz uma análise do fluxo estabelecido pelo projeto do usuário. Nessa análise são verificadas as compatibilidades de conexões, parâmetros e recursos do dispositivo alvo (*target*). O gerador faz uma varredura componente-por-componente, iniciando-se através dos componentes de aquisição. Durante a varredura podem ser detectados 4 tipos de interconexões (Figura 46) :

Conexões software <-> software: Geralmente acarretam na construção de chamada de procedimento no projeto Software;

Conexões Software -> Hardware: Dependendo do componente em análise, acarretar na criação de rotinas de comunicação via PCI-express, criação de blocos de código para instruções customizadas, criação de conexão via barramento Avalon;

Conexões Hardware -> Software: Pode acarretar na criação de rotinas de monitoramento do barramento PCI-express ou comunicação com barramento Avalon;

Conexões Hardware <-> Hardware: Podendo resultar na criação de FIFOs, Wires, Registradores, Acesso a memória SDRAM ou acesso ao barramento Avalon.

Figura 45 – Interligação entre IPs *glue-box*

Fonte: Elaborada pelo autor.

Muitas dessas conexões podem não estar completamente implementadas, sendo necessário a interação do usuário no preparo para a execução no FPGA. O gerador de projetos cria uma pasta específica para cada tipo de IP, copiando os arquivos de template para o diretório.

Uma das partes trabalhosas da construção de um IP é adicionar a capacidade de geração automatizada de código fonte pois, pode haver grande variação de acordo com a forma que o IP é conectado e parametrizado.

No *framework*, um mesmo algoritmo pode ter diversas apresentações. Por exemplo, um IP para conversão RGB -> Cinza pode ser implementado: pelo Bluespec, gerando um código de verilog específico; Implementado manualmente na linguagem Verilog (pelo usuário); Implementado para executar no processador ATOM através de bibliotecas OpenCV, ou de C++ padrão; Implementação para Software Embarcado para o processador NIOS II; Implementação como uma Instrução Customizada. Além disso, a conversão de RGB para a escala de cinza pode apresentar até 7 algoritmos diferentes de implementação, multiplicando a quantidade de arquivos diferentes de código. A Figura 46 ilustra um exemplo dessa característica. Nota-se também, que o arquivo resposta gerado por cada algoritmo e de cada arquitetura pode apresentar diferenças, pois depende dos recursos da arquitetura onde foi implementado, por exemplo a diferença na precisão das casas decimais, a margem de erro e as adaptações heurísticas para suportar uma arquitetura diferente.

4.4.1 Validações de Núcleos de Propriedade Intelectual

Para estender a ferramenta com o novos IP que possam ser embarcados, é necessário o desenvolvimento de estruturas de código em um formato padrão. Para garantir que os resultados obtidos na simulação sejam próximos aos resultados práticos, são exigidos alguns cuidados.

Figura 46 – Diversas apresentações de um IP

Fonte: Elaborada pelo autor.

O suporte ao NIOS II C++ depende da cadeia de ferramentas GCC ([NIOSII, 2017](#)). A cadeia de ferramentas Nios II GCC 4 C++ suporta os seguintes recursos: Polimorfismo; *Friendship* e herança; Herança múltipla; Classes virtuais base; *Friendship*; O qualificador do tipo *mutable*; Namespaces; *Templates*; Alocação de memória dinâmica de estilo novo e exclusão; Sobrecarga do operador; Biblioteca de modelos padrão (STL); Exceções e conversões dinâmicas de estilo em novo não são suportados.

Para a validação de IP de software ([Figura 47](#)), deve-se usar um padrão pré-estabelecido de construção. O arquivo principal “IP.c” contém o algoritmo proposto e deve ser compatível com as bibliotecas utilizadas pelo NIOS II. Recomenda-se executar o script de teste “TestBench.c”. Isto possibilita ler as entradas de dados de um arquivo de origem “iStream.bin” para ser usado no teste de algoritmo e salvar a saída para um arquivo de verificação “oStream.out”. Finalmente, com o sistema em execução no ambiente embarcado, um profiler é realizado para extrair várias características do algoritmo proposto.

O BlueSim permite a validação do hardware gerado através de um programa escrito na própria linguagem “TestBench.bsv”. Assim como os blocos validados em software, as entradas de dados podem ser lidas a partir de uma fonte de arquivo “iStream.bin” e a saída gerada em um arquivo de verificação “oStream.out”. Em ambos os casos, pode-se comparar os arquivos “fStream.bin” com a saída da implementação em software do PC.

Para a validação de IP em hardware (Figura 48), o arquivo principal “IP.bsv” contém o algoritmo proposto em linguagem Bluespec. Caso necessário, a linguagem permite a reutilização de estruturas em Verilog usando um encapsulamento para essa finalidade. Uma vez compilado, gera-se um arquivo Verilog “IP.v” que pode ser adicionada a um template de validação no Quartus II. Através de técnicas de profiler é possível extrair várias características do hardware gerado.

Figura 47 – Validação de um IP em Software

Fonte: Elaborada pelo autor.

Figura 48 – Validação de um IP em Hardware

Fonte: Elaborada pelo autor.

RESULTADOS

Dentre os Sistemas Avançados de Auxílio ao Motorista baseados em câmera, este capítulo apresenta um estudo de caso para sistemas de detecção de pedestres. Para a avaliação, foram implementados dois projetos utilizando o *framework* ADAS-Vision. Nestes projetos, diversas personalizações foram realizadas demonstrando a exploração do espaço de projetos (DSE). Todos os modelos foram executados através da placa de desenvolvimento Terasic DE2i-150 que utiliza um processador Intel ATOM Cedarview N2600 e um FPGA Altera Cyclone IV EP4CGX150DF31.

O primeiro projeto utiliza a subtração sucessiva de quadros de imagem para fazer a detecção de movimento, para isso é necessário aplicar um limiar de intensidade onde são detectados os pontos da imagem que correspondem às bordas dos objetos em movimento. Através desses pontos, a segmentação é feita pela identificação de aglomerados de pontos através do algoritmo para identificação de componentes conectados (*connected-component labeling*). Com objetivo de eliminar pontos desconexos, é aplicado na etapa anterior, um filtro de processamento de imagem de erosão. Os objetos segmentados pelo *labeling* são então, filtrados por um algoritmo de medida de proporção construído a partir da pesquisa de [Dollar et al. \(2012\)](#). Finalmente, o classificador seleciona os objetos que estão na região de interesse, delimitada pela direção do veículo.

Duas sequências de imagens por vídeo de um ambiente real foram utilizadas neste teste, totalizando 46230 quadros. O vídeo apresenta um pedestre com vestimentas com cores contrastantes que passa na frente do veículo em dois cenários diferentes. O veículo utilizado no teste, locomove-se à velocidade de 30 km/h. A construção deste projeto baseou-se no conceito da implementação do trabalho de [Martinez \(2011\)](#). Para adaptação ao ADAS-Vision e da placa de desenvolvimento de FPGA (usada nos testes neste trabalho), foi necessário a reconstrução e adaptação dos blocos originais, partindo dos mesmos conceitos das heurísticas e algoritmos para a conversão em IPs de Hardware e IPs de Software.

O segundo projeto faz a detecção de pedestres através de Maquinas de Vetores de Suporte (SVM) classificando um vetor do descritor de Histograma de Gradiente Orientado (HoG). O descritor foi previamente treinado para detectar pessoas. Com objetivo de comparar os resultados obtidos com os principais rankings de detecção de pedestres, o sistema fez a exportação dos dados de resposta de detecção para um arquivo MatLab, no padrão estabelecido pela *Caltech University*.

Uma das mais populares bases de *benchmark* para detecção de pedestres pertence à Caltech-USA (DOLLAR *et al.*, 2012), neste segundo projeto foi utilizado esta base apenas com o Set006 com 18 sequências de vídeos. Cada sequência foi convertida em imagem totalizando 34.855 arquivos de quadros. As filmagens foram feitas por uma câmera (trabalhando em 30Hz) em um veículo trafegando pelas ruas de Los Angeles, nos Estados Unidos.

5.1 Exploração do Espaço de Projetos (DSE)

Inicialmente, como tema desta tese, foram considerando IPs comuns para sistemas ADAS e especialmente para sistemas de detecção de pedestres. Apesar de diversas abordagens para a implementação desses sistemas ¹, ainda existem diversos algoritmos que podem ser combinados e otimizados para melhorar os resultados.

As principais fases de diversos sistemas ADAS para prevenção de acidentes voltados para detecção de pedestres podem ser resumidos a três principais fases:

1. **Pré-processamento de imagem:** Dentre os IP para pré-processamento de imagens destacamos os filtros 2d, detecção de bordas, limiar, suavização, *emboss* (em relevo), tratamento de cor, transformações morfológicas, transformação *Hough Line* e marcação;
2. **Extração de características:** Destacam-se: HOG (*Histogram of Oriented Gradients*), SIFT (*Scale-Invariant Feature Transform*), detecção de canto, HAAR, LBP (*Local Binary Patterns*);
3. **Classificação:** *Template Matching*, KNN (*k-Nearest Neighbors*), SVM (*Support Vector Machines*), AdaBoost (*Adaptive Boosting*), Redes Neurais Artificiais (ANN), HAAR Cascade, ID3 (Árvore de Decisão), *Random Forests*, *Naive Bayes*.

Além dessas três etapas e complementando os Sistema de Proteção a Pedestres (PPS), existe uma última etapa responsável pela interpretação dos dados de cenário. As principais funções desta etapa estão direcionadas para determinar uma possível rota de colisão, determinação da distância do pedestre e situações de risco.

¹ A lista de referência dos diversos trabalhos voltados para detecção de pedestres pode ser encontrada online em <http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/files/algorithms.pdf> (acessado em junho de 2017)

Tabela 1 – Análise combinatória para diversos modelos de projetos

Etapa	IPs Possíveis HW+SW	Exemplo de uso	Possíveis combinações pelo número de IPs				Combinações
			4	3	2	1	
A	136	de 1 a 4	327211920	2460240	18360	136	329.690.656
B	24	de 1 a 3		12144	552	24	12.720
C	26	de 1 a 2			650	26	676

Quantidade de projetos possíveis (A x B x C): 2.834.917.637.560.320

Fonte: Elaborada pelo autor.

Levando-se em consideração os IPs mais comuns para este estudo de caso, pode-se contabilizar, apenas para a etapa de pré-processamento, cerca de 68 diferentes implementações de algoritmos em software (IPs Software). Neste contexto, estamos considera-se que cada implementação desses respectivos IPs, pode ter a sua versão em Hardware (IPs Hardware) e assim totalizando 136 IPs (Hardware e Software). Desta mesma forma (somando-se os dois tipos de IPs), obtém-se para a fase de extração de características mais 24 IPs e, por fim, na fase de classificação, somam-se 26 IPs. A figura [Figura 49](#) mostra um diagrama com os diversos algoritmos que podem ser utilizados para a construção de sistemas para detecção de pedestres e outros sistemas.

Através dessas quantidades de IPs, é possível inferir uma análise combinatória do número de possíveis projetos que podem ser construídos dentro do ambiente do ADAS-Vision permitindo uma ampla exploração do espaço de projetos. A [Tabela 1](#), demonstra de forma simplificada, exemplos hipotéticos de construção de sistemas, variando-se na etapa A (pré-processamento) a utilização de 1 a 4 tipo de diferentes IPs, para a etapa B (extração de características) exemplos de utilização entre 1 a 3 IPs e, por fim, de 1 a 2 IPs na etapa C (classificação).

Nesse exemplo, ainda é importante destacar que, não estão sendo contabilizadas todas as possíveis variações dos parâmetros do IP (cada implementação de IP pode permitir diversas personalizações de atributos pelo usuário da ferramenta). Da mesma forma, também para simplificar a análise, foi considerado que não haveria repetição do mesmo IP no mesmo projeto (que em casos reais pode ocorrer). A ordem em que os IPs são sequenciados no projeto, pode alterar significativamente o resultado final. Diversas dessas combinações também podem apresentar resultados inválidos mas, em contra partida, observa-se que diversas novas combinações podem ser exploradas a fim de obter novos resultados ainda inexplorados.

5.1.1 Método de Exploração do Espaço de Projetos

O processo de Exploração do Espaço de Projetos (DSE), esboçado na [Figura 50](#), tem o objetivo de maximizar as opções de respostas, buscando melhores resultados para projeto. Para isso, são selecionados os conjuntos combinados de componentes IPs e parametrizações que apresentem soluções factíveis. Este conjunto, criado para executar lógicas computacionais específicas e com requisitos restritivos, é denominado neste trabalho pelo termo configuração de projeto. Os requisitos definem os limites para a área de exploração do projeto, por exemplo, número máximo de elementos lógicos, o consumo elétrico máximo (potência), o tempo de mínimo resposta, a precisão dos resultados e outros diversos, que dependem de cada projeto. Dessa forma, limita-se o número e os tipos de recursos, que um sistema pode conter e também define os objetivos específicos para a exploração do espaço de projetos.

O processo de exploração das configurações pode ser dividido em duas fases: otimização de parâmetros e otimização de recursos. A fase de otimização de parâmetros modifica as propriedades que cada IP, permitindo mudar algoritmos ou alterar as características de funcionamento que influenciam no resultado. Nesta fase, através da simulação, é possível verificar se a configuração do projeto está totalmente implementada e tem resultados satisfatórios de resposta. Para não limitar a arquitetura alvo, ou para melhorar os atributos do sistema, a fase de otimização de recursos modifica a arquitetura no nível de componentes (IPs), adicionando e/ou removendo unidades. Todas as alterações dessa configuração ou parametrizações, resulta em uma relação custo / benefício diferente. As configurações mais interessantes são tais que, resultam em melhores atributos de qualidade ([MANTYNEVA, 2009](#)), mantendo-se dentro dos requisitos específicos. Estes pontos de interesse são chamados de Pontos de Pareto.

Além dos atributos quantitativos, o trabalho dos autores [Oliveira et al. \(2013\)](#) estabelece um Modelo de Qualidade (QM) para avaliar arquiteturas de software específicas para o contexto de ADAS. Neste estudo, os autores dedicaram-se às arquiteturas dos Sistemas Embarcados Automotivos. Posteriormente, o trabalho de [Oliveira \(2017\)](#) estabelece características e atributos para os requisitos de qualidade direcionados aos Sistemas Embarcados Críticos (CES), onde complementa também, os requisitos para sistemas automotivos. Em sua conclusão, evidencia-se que os modelos de qualidade ainda são complexos e precisam ser melhor explorados. Na construção de um projeto dentro do ADAS-Vision, diversos desses atributos apresentados na [Tabela 2](#) podem ser levados em consideração para a Exploração do Espaço de Projetos, trazendo ao usuário da ferramenta, melhores condições para uma tomada de decisão.

5.2 Resultados do Primeiro Projeto

Neste primeiro projeto, avaliou-se o desempenho e a precisão alcançado pelo modelo de implementação. A principal modificação para contribuição nos resultados foi a substituição do estabilizador de vídeo original por um estabilizador que usa algoritmo de fluxo óptico. O

Tabela 2 – Tabela de atributos para Modelo Qualidade (QM) para Sistemas Embarcados Críticos (CES)

Característica	Sub-Característica
Confiabilidade	Disponibilidade
	Tolerância à falha
	Maturidade
	Recuperabilidade
Eficiência de desempenho	Capacidade
	Utilização de recursos
	Comportamento do tempo
Usabilidade	Capacidade de aprendizado
	Operabilidade
	Proteção contra erros de usuário
Manutenção	Analisabilidade
	Modificabilidade
	Modularidade
	Reusabilidade
	Testabilidade
Segurança	Autenticidade
	Confidencialidade
	Integridade
	Não-repúdio
Compatibilidade	Coexistência
	Interoperabilidade
Portabilidade	Adaptabilidade
Satisfação	Confiabilidade
Análise de Risco	Atenuação de risco econômico
	Atenuação de risco ambiental
	Atenuação de risco de saúde e segurança

Fonte: [Mantyneva \(2009\)](#), Adaptado).

Figura 50 – Processo de exploração do espaço de projetos

Fonte: Oliveira (2017, Adaptado).

conjunto de resultados obtidos nos testes ateu-se às medidas de tempo de execução e da precisão da detecção. Explorações foram obtidas variando-se os filtros, parâmetros de IPs e o tipo de estabilização de vídeo, como citado anteriormente.

Neste estudo de caso, o processo completo de elaboração está representado pela Figura 51. Na primeira fase (Aquisição), a base de imagens em formato de arquivo de vídeo é carregada, em seguida a imagem é ajustada para permanecer estável (*deshaker*) e passa por um filtro de ruídos. O processamento faz a detecção de movimento e classifica os objetos detectados pela segmentação da imagem, em seguida, os dados são enviados pela saída de vídeo FPGA passando pelo barramento PCI-express.

Este projeto faz a subtração sucessiva de quadros para detecção de movimento, dessa

forma, pequenas variações na altura ou no ângulo da câmera, ocasionam detecções incorretas. A utilização do estabilizador por fluxo óptico fez com que o sistema apresentasse uma melhora de 8% na detecção de pedestres quando comparado com o projeto original. Em consequência, o novo método necessita de mais recursos computacionais que o anterior.

Figura 51 – Estudo de caso: Projeto 1

Fonte: Elaborada pelo autor.

A construção do projeto no Ambiente de Desenvolvimento Integrado (IDE - *Integrated*

Development Environment) do framework ADAS-Vision iniciou-se com a inserção do IP para fornecimento dos dados de vídeo (*SW Video Source*).

Os dados de cada quadro do vídeo foram inseridos em um IP para detecção de movimentos de quadros sucessivos. Este IP emite uma coordenada respectiva ao deslocamento vertical, fornecendo dados para o componente de ajuste de altura (*SW Affine Transformation*). Como os dois componentes necessitam da mesma imagem de entrada, um IP de clonagem da imagem (*SW Image Clone*) foi inserido anteriormente às suas entradas. Após esta modelagem, o IP de ajuste de altura fornece a imagem estabilizada para o IP seguinte.

O processo de estabilização de imagem iniciou-se com a conversão da imagem estabilizada colorida (RGB) para a escala de cinza (*HW Convert Gray Scale*). Em seguida, após a aplicação de um filtro de erosão (*SW Erosion Filter*), a imagem é inserida no IP de subtração de quadros consecutivos (*SW Background Subtraction*). Como resultado, a saída apresenta uma máscara contendo apenas os pontos referentes a subtração dos quadros, destacando apenas os pontos onde existem diferenças que, neste caso, associa-se à movimentação de todo cenário capturado na filmagem pela câmera em relação ao veículo.

O trabalho de Dollar *et al.* (2012) avalia que, estatisticamente, a proporção entre altura e largura do pedestre é aproximadamente: $Largura = 0,41 \times Altura$. Dessa forma, a classificação dos pedestres (*IP SW Simple Classifier*) foi realizada através do filtro de proporção entre altura e largura dos objetos segmentados. Esta segmentação foi feita através de um IP que identifica componentes conectados (*SW Blob Detection*), disponibilizando um vetor de coordenadas de retângulos para o IP de classificação.

A Figura 52 mostra a interface (IDE) que apresenta esse modelo na aplicação ADAS-Vision. Para testes ou validações, alguns IPs foram substituídos por seus equivalentes em hardware, como exemplo na figura o IP de conversão para escala de cinza (*HW Convert Gray Scale*) e o componente para envio da imagem ao FPGA através do barramento *PCI-express* (*IO PCI-e Send FPGA*).

Durante o processo de construção do projeto, foi possível visualizar, ler e capturar os resultados, imediatamente após a interligação dos IPs. Ao fim da execução da montagem, foi possível ter uma ampla ideia do resultado final e fazer ajustes em parâmetros, a fim de corrigir ou obter, melhores resultados. Por fim, para a execução na placa DE2i-150, foi realizada a geração do projeto (opção *Build* da ferramenta). Esta opção, cria na pasta do projeto, um subdiretório com os arquivos fontes de um novo projeto (escrito em C++), relativos somente aos códigos dos IPs utilizados no projeto. Também foram inseridos contadores de tempo entre as principais rotinas para análise posterior. Uma vez gerado o projeto, este foi compilado para que pudesse ser executado no ambiente de testes (placa de desenvolvimento FPGA). Apesar do ambiente de simulação permitir extrair dados de resultados diretamente na placa de desenvolvimento, a compilação se faz necessária para isolar as possíveis interferências no desempenho da aplicação gerada pela adição de diversos componentes de controle.

Figura 52 – Projeto 1 em simulação na ferramenta

Fonte: Elaborada pelo autor.

5.3 Resultados do Segundo Projeto

O segundo projeto implementa a detecção de pedestres através do classificador SVM para um um descritor de características HoG, utilizando-se bibliotecas baseadas no trabalho de [Dalal e Triggs \(2005\)](#). Na avaliação, a exploração ficou direcionada à utilização de filtros de entrada para avaliar a influência na precisão da detecção. Conforme o trabalho de [Wang et al. \(2013\)](#) a alteração de contraste ou espaço de cores pode influenciar diretamente no resultado da detecção.

Para validar a detecção, os dados resultantes da detecção de todos os quadros do conjunto de testes, foram exportados para o formato padrão do *benchmark* da *Caltech University*. Em seguida, utilizou-se o programa em MatLab também fornecido pela universidade que fez a leitura e posteriormente a representação gráfica ([Figura 53](#)) que permite comparar o resultado do projeto implementado no ADAS-Vision com outras aplicações cientificamente já conhecidas e que usam a mesma base de testes.

Figura 53 – Resultado comparativo: Projeto 2

Fonte: Elaborada pelo autor.

O processo completo de elaboração está representado pela [Figura 54](#). Na primeira fase (Aquisição), a base de imagens em formato de arquivo (.jpg) é carregada, em seguida a imagem é ajustada para a escala apropriada e convertida para escala de cinza. O processamento de detecção é feito através de um classificador que exporta os dados para o formato do software MatLab.

Figura 54 – Estudo de caso: Projeto 2

Fonte: Elaborada pelo autor.

A construção deste projeto no Ambiente de Desenvolvimento Integrado (IDE - *Integrated Development Environment*) do framework ADAS-Vision iniciou-se com a inserção do IP para carregamento dos dados de imagem sequencial em diretório apropriado (*SW Image Source*).

Os dados de cada quadro do vídeo foram enviados para um IP de conversão de RGB para Escala de cinza (*SW Convert Gray Scale*). Esta IP reduz as três dimensões de cores para uma

única, reduzindo a quantidade de dados transmitidos para o próximo componente.

Para padronização de tamanho de quadros, foi utilizado um IP de ajuste de escala (*SW Resize*) que usa por padrão interpolação bilinear. Os quadros foram ajustados para a proporção de 640 x 480 pixels.

O IP descritor de características HoG multiescalas (*SW HoG Descriptor*) usa a imagem de entrada para criar um vetor das características extraídas de cada pixel da imagem. Esse vetor é enviado juntamente com a imagem original para o IP de classificação SVM (*SW SVM Classifier*) que conclui a detecção dos pedestres na imagem.

A [Figura 55](#) mostra a interface (IDE) que apresenta esse modelo na aplicação ADAS-Vision. Para testes ou validações, alguns IPs tiveram seu parâmetros modificados com objetivo de melhorar a precisão da detecção.

Pela característica padrão do ambiente, foi possível visualizar, ler e capturar os resultados, imediatamente após a interligação dos IPs. e posteriormente, foi realizada a compilação do projeto desenvolvido (através da opção no menu Build), criando na pasta do projeto, um subdiretório com os arquivos fontes de um novo projeto (escrito em C++) e executado no ambiente de testes (placa de desenvolvimento FPGA).

Na [Figura 56](#) apresenta-se um gráfico com as dependências de bibliotecas utilizadas no projeto. Nota-se que, nesta implementação, a maioria das referências do projeto é externa. Reutilizando-se assim, bibliotecas de softwares pré-existentis para o uso em processamento da imagens.

Uma possível melhoria desta implementação encontra-se no trabalho de [Rettkowski, Boutros e Göhringer \(2017\)](#). Nesta pesquisa, os autores apresentam três projetos diferentes utilizando o algoritmo HOG. Primeiramente com a implementação em software, são realizadas otimizações com paralelismo, através da biblioteca de software OpenCV, utilizando como plataforma de testes, um processador ARM dual-core. Através da análise de gargalos no tempo de processamento, um coprojeto de Hardware e Software (do algoritmo HOG original), é implementado. Para isso, utilizou-se uma Linguagem de Síntese de Alto-Nível (HLS) para sintetizar as funções codificadas por software para o FPGA, gerando assim, as interfaces de dados para intercomunicação entre o processador e o FPGA. Por fim, foi realizada uma implementação completa baseada em FPGA, com pipeline do algoritmo HOG, juntamente com um classificador AdaBoost.

Figura 55 – Projeto 2 em simulação na ferramenta

Fonte: Elaborada pelo autor.

Figura 56 – Projeto 2 análise de dependências

Fonte: Elaborada pelo autor.

5.4 Discussão sobre os resultados obtidos

O primeiro projeto mostrou desvantagens por ser um modelo bastante limitado pois, é sensível às variações de cores do asfalto, vibrações da câmera e luminosidade causando erros de detecção. Também não faz a detecção de pedestres em regiões potenciais de risco, é sensível apenas ao que estiver à frente do veículo denominada Região de Interesse (ROI). Apesar disso, esta abordagem por ter complexidade relativamente menor que os detectores atuais mais utilizados, apresenta uma maior velocidade de resposta. Outra vantagem, é permitir detectar outros objetos que estejam na direção do veículo, para isto, são necessários apenas ajustes no filtro de proporção.

O segundo projeto é especializado no reconhecimento de padrões previamente treinados para detecção de pedestres, contudo, a quantidade de processamento é maior do que o primeiro projeto. A precisão para maiores distâncias também diminui, isso ocorre devido à diminuição do tamanho do pedestre (em pixels) na imagem.

Os projetos passaram por diversos ajustes durante sua montagem, ambos permitem a adição ou alteração de IPs possibilitando uma infinidade de customizações. Cada IP do sistema possui características próprias de sua arquitetura, permitindo que tais características se tornem métricas para uma tomada de decisão das varias hipóteses de combinações. Tentar construir todas as combinações ou simplesmente escolher aleatoriamente torna-se inviável pela quantidade de combinações já exemplificadas anteriormente.

Os IPs de Hardware apresentam características que podem variar com seus parâmetros. Dentro do IDE do ADAS-Vision, o IP Hardware passa por uma recompilação toda vez que seus atributos mudam e assim, alterando os requisitos mínimos para sua utilização em FPGA. Dentre as características podemos extrair: Número de Elementos lógicos, Velocidade de Clock, tempo de resposta (latência e *throughput*), Consumo, Recursos como Multiplicadores embarcados, Memória, pinos de Entrada e Saída, *Phase-Locked Loop* (PLLs), Processadores de Sinal Digital (DSP), registradores, entre outros.

No caso dos IPs de Software para SoPC, os recursos do algoritmo do IP ou também suas parametrizações, podem ocasionar a necessidade de mudanças de arquitetura do processadores Softcore, como por exemplo a necessidade de utilização de ponto flutuante, capacidade de memória e diversos outros recursos integrados que também alteram os requisitos mínimos de uso no FPGA. Além disso, os softwares podem ser analisados para extração de diversas características como: tempo de resposta (latência e *throughput*), linhas de código, complexidade, memória necessária, dentre outras.

Além disso, ambos (IPs Hardware e IPs Software) podem apresentar diversas características resultantes da execução de seus algoritmos como, por exemplo a precisão do resultado ou a margem de erro.

Com todas essas características, as quais podem ser extraídas dos IPs implementados no

ADAS-Vision e também com os requisitos qualitativos (apresentados anteriormente pelo modelo de qualidade), gera-se uma demanda para um sistema de tomada de decisão.

A [Figura 57](#) mostra uma planilha com um exemplo simplificado de extração de características de oito projetos diferentes de detecção de pedestres. Neste exemplo, foram considerados apenas alguns requisitos de seleção: o custo do Circuito Integrado (CI), o Consumo Estimado do projeto, a Velocidade de Resposta da detecção em quadros por segundo e a Precisão da Detecção.

Os recursos necessários somados de cada IP (Hardware ou Software) são representados por Elementos Lógicos, Memória RAM, Velocidade do Clock, Número de PLLs e Número de Multiplicadores embarcados. Esses recursos inicialmente acarretam na escolha de um Circuito Integrado (FPGA ou Processador) de acordo com cada projeto. Para isso, a coluna modelo apresenta um código de referencia do CI que mais se aproxima desses requisitos mínimos. Esses CIs, implicam em uma parte do custo de um projeto, apresentado na coluna apropriada.

Figura 57 – Planilha para tomada de decisão

Projeto	Precisão	Desempenho		Financeiro	Recursos Necessários				
	Precisão Detecção	Velocidade (Frames/s)	Consumo (Watts x 10 ⁻³)	Custo C.I. (US\$)	Modelo C.I.	Elementos Lógicos	RAM (Mbytes)	Clock (MHz)	PLLs Multiplic.
1	95%	20	250	42,00	Atom N2600		2000	1600	
2	70%	15	200	36,40	EP4CE15M8I7N	4515	2000	100	3 5
3	70%	30	250	156,40	EP2C35F484C6N	25325	4000	80	4 10
4	83%	30	210	36,40	EP4CE15M8I7N	4910	2000	90	3 6
5	90%	33	215	82,80	EP1C20F324C7N	9244	1000	120	3 8
6	82%	34	220	36,40	EP4CE15M8I7N	5241	2000	125	2 10
7	85%	85	150	968,00	EP3CLS150F780C8N	112548	4000	150	1 20
8	90%	95	110	798,60	EP3CLS100U484I7	95241	2000	170	2 15
9	83%	90	120	968,00	EP3CLS150F780C8N	105246	2000	160	1 10

Fonte: Elaborada pelo autor.

Neste exemplo, os oito modelos foram divididos em três diferentes tipos de implementação: sistemas totalmente em Hardware, sistemas totalmente em Software e os sistemas em coprojeto. Os indicadores distribuídos com os pesos respectivos, representam os custos qualitativos associados ao projeto. Neste caso, foi atribuída uma maior relevância ao valor do custo financeiro associado, em seguida, pela precisão de resposta e por fim, o desempenho dos projetos apresentados. Na métrica apresentada, quanto maior o índice, pior é a avaliação do resultado.

O **projeto 1** foi implementado totalmente em software executando em um processador Intel ATOM, nota-se que o projeto apresentou a melhor precisão na detecção, contudo, a implementação apresenta um consumo elevado para execução. O **projeto 2** representa a execução do sistema em um processador embarcado (SoPC), neste caso, apesar do seu baixo custo, a desvantagem se encontra na baixa taxa de precisão. O **projeto 3** é a tentativa de paralelização do código em vários núcleos de processamento, obtendo uma melhora significativa na velocidade de resposta. O **projeto 4** utiliza Instruções Customizadas no SoPC (SoPC + C.I.), melhorando a precisão e a velocidade de resposta em relação ao segundo projeto. O **projeto 5** faz, além do uso de instruções customizadas, parte do processamento em blocos de Hardware, melhorando significativamente a precisão. O **projeto 6** transfere toda a execução das instruções customizadas para execução direta em hardware, em consequência diminuindo o custo do FPGA., contudo, essa abordagem teve resultado de precisão inferior ao projeto anterior. O **projeto 7** apresenta a solução implementada totalmente em FPGA, em consequência, o número de elementos lógicos necessários, elevou o custo do FPGA necessário. Para minimizar o problema do projeto anterior, o **projeto 8** utiliza recursos de Processadores de Sinais Digitais (DSP) para melhorar o desempenho e reduzir o custo. Por fim, o **projeto 9** faz a implementação total em hardware mas usando funções otimizadas pelo fabricante do FPGA, melhorando a velocidade de resposta em relação a construção somente com funções próprias.

Observa-se que, com poucos atributos e também com poucas combinações, existem diversos índices que podem ser levados em consideração para a melhor tomada de decisões. Neste exemplo, cada projeto poderia ainda ter ajustes de parametrizações de IPs que acarretariam em mais opções de escolha. A [Figura 58](#) mostra um exemplo simplificado da Árvore de Decisão aplica para a planilha apresentada. Nesta figura, nota-se pelo **nó 1** que houve a decisão de usar o segmento do **nó A** ao invés da solução totalmente em Hardware. O fator que mais contribuiu para isso foi o custo elevado dos FPGAs necessários. A decisão entre uma solução totalmente em software perdeu por pouca diferença para a solução em coprojeto. Neste caso, o que mais contribuiu para isso foi o baixo desempenho apresentado pela primeira abordagem. No resultado final, a melhor solução seguindo os requisitos selecionados, foi o **projeto 5**, onde faz o uso de recursos de software embarcado com instruções customizadas e também rotinas em Hardware executando no FPGA.

Figura 58 – Arvore de decisão

Fonte: Elaborada pelo autor.

Através da Curva de Pareto traçada pela somatória de cada atributo de custo de projeto (Figura 59), observa-se que, nesse conjunto em avaliação, mais de 60% dos custos estão concentrados em apenas dois atributos: consumo e valor financeiro. Entretanto, esses valores estão diretamente associados aos pesos escolhidos. Esses pesos, foram determinados qualitativamente na fase de projeto para determinar a relevância dada a cada atributo, influenciando diretamente na tomada de decisão.

Figura 59 – Gráfico de Pareto

Fonte: Elaborada pelo autor.

Esse tipo de gráfico pode ser aplicado a diversos atributos dependendo da escolha do projetista. Uma das vantagens de sua utilização é a facilidade de encontrar os atributos que representam maior peso para a maioria dos projetos, permitindo assim, determinar os maiores fatores de influência.

Deste modo, procurou-se demonstrar com estes exemplos (totalmente implementados pela ferramenta ADAs-Vision) toda a operacionalidade, flexibilidade e potencialidade da ferramenta para a Exploração do Espaço de Projeto (DSE). Novos exemplos (e novos IPs) estão sendo construídos para tornar a ferramenta mais abrangente na área de processamento de imagens e num futuro próximo os mesmos serão incorporados a ela.

CONCLUSÕES

6.1 Aspectos Gerais

O trabalho fez uma discussão geral sobre sistemas avançados de assistência ao motorista baseados em câmeras e na computação reconfigurável, nesse sentido procurou-se ressaltar os requisitos fundamentais de hardware e software para suportar o modelo de implementação e sua modelagem.

No projeto e construção do *framework* ADAS-Vision, procurou-se atender algumas questões básicas tais como: flexibilidade na elaboração dos projetos, velocidade no tempo de desenvolvimento, exploração de espaço de projeto, reuso de IPs, simulação e finalmente a implementação do coprojeto de hardware e software nas placas de FPGA disponíveis no LCR (Laboratório de Computação Reconfigurável). Essas premissas foram atendidas, viabilizando a aplicação do *framework* também para técnicas de ensino de várias disciplinas de sistemas embarcados no ICMC-USP. Em função da sua grande importância, a detecção de pedestres foi enfatizada através de uma abordagem mais detalhada. Cade ressaltar que o sistema está preparado para detectar qualquer objeto que se mova na imagem, dando-se preferência aos pedestres mas não excluindo a possibilidade de detecção de outros objetos (carros, bicicletas, placas de trânsito, detecção de mudança de pista, etc). Todas essas funcionalidades somente são possíveis devido multiblicidade da biblioteca de IPs que implementam diversas técnicas de processamento de imagem.

6.2 Contribuições do Trabalho

O principal resultado desta pesquisa é a construção do *framework* ADAS-Vision em si, possibilitando a exploração do espaço de projeto e diminuição do esforço de desenvolvimento, visando um ganho na velocidade de produção de sistemas ADAS baseados em visão. É previsível

que exista um maior esforço inicial na construção de novos IPs, dos gabaritos e dos componentes de softwares. Contudo, posteriormente, um usuário poderá usufruir destes novos blocos e construir vários sistemas ADAS de uma maneira muito mais eficiente e que leve vantagem em relação ao fluxo de engenharia normal de projeto.

A inovação desta pesquisa apresenta-se sob a forma de uma nova abordagem para elevar o nível de abstração de aplicações de coprojeto de hardware/software para sistemas embarcados baseados em computação reconfigurável. A principal contribuição é a formulação de um *framework* para a geração automática de hardware e software embarcado, através de uma interface visual, específica para o desenvolvimento de sistemas de ADAS com visão e de fácil programabilidade para o usuário. A parte crítica para o sucesso do projeto encontra-se na construção do sistema de intercomunicação entre os diversos blocos de IP e os componentes de software, abstraindo do usuário final inúmeros detalhes de hardware, tais como gerenciamento de memória (on-chip, off-chip), interrupções, cache, tipos de dados (ponto flutuante, ponto fixo, inteiros) e etc. Com o *framework* também é possível um uso mais amigável da tecnologia de FPGA que carece de ferramentas intuitivas que auxiliem na sua programação, além disso, o *framework* tem a vantagem de retornar resultados imediatos da simulação, sem a necessidade da síntese que pode levar elevado tempo de processamento.

A partir dos dados apresentados no [Capítulo 5](#), nota-se que no esquema de detecção de pedestres implementado, a estabilização de imagem usando o fluxo óptico é um fator crítico para melhores resultados. Observa-se também, que o vetor para o algoritmo de estabilização, apesar da sua simplicidade, traz bons resultados em relação a resultados sem estabilização de imagem. Neste trabalho, utilizando-se de componentes baseados em outros projetos e com as mesmas amostras de fontes de dados foi possível obter os mesmos resultados originais, mostrando assim, que é possível migrar projetos externos para a ferramenta sem criar interferência nos resultados e permitindo a exploração do espaço de projetos mais sofisticados.

Como principal desvantagem destacou-se o tempo total de integração e de simulação do projeto definido pelo usuário. A cosimulação envolvendo diversas ferramentas de diversos fabricantes acumula um tempo total para síntese relativamente oneroso. Uma maneira de se diminuir esse tempo total de integração pode ser explorado com o teste de outras ferramentas de simulação de hardware disponíveis, tais como, HDL Coder (Matlab) e OpenCL (Intel).

A construção do ambiente de simulação possibilitou a validação comportamental dos sistemas ADAS pré-definidos anteriormente. O *framework* serviu como uma prova de conceito de que a exploração do espaço de projeto é possível e necessária devido a inúmeros cenários existentes no contexto de ADAS.

Um ponto chave do sistema a ser explorado é a ampliação da biblioteca de IPs e de novos gabaritos envolvendo as diversas condições de trânsito e ambientais, tais como condições de baixa visibilidade, condução noturna, etc.

Em relação ao software desenvolvido do framework pode-se afirmar que foram aplicados conhecimentos de diversas áreas da engenharia de software e linguagens de programação, tendo sido desenvolvido integralmente pelo autor deste trabalho. O ambiente foi desenvolvido utilizando-se programação C++ para multi-plataformas, bibliotecas de processamento de imagens (OpenCV), XML, CUDA, técnicas de aprendizado de máquinas, linguagens de descrição de hardware, técnicas de bases de dados, compiladores, ferramentas de simulação de hardware, bibliotecas aceleradoras de software, ferramentas de sínteses da Altera e integrações com Matlab.

Desta forma, foram destacados e discutidos os aspectos principais para que o framework ADAS-Vision possa ser utilizado no desenvolvimento de sistemas ADAS e no ensino de sistemas embarcados, envolvendo diversas disciplinas da graduação e pós graduação (Coprojeto de hardware e software, Projeto e Implementação de Sistemas Embarcados, etc). Durante a elaboração deste trabalho, não foi constatado na pesquisa bibliográfica ferramenta semelhante a este framework, no sentido de integrar completamente hardware e software em um único ambiente de simulação e exploração de espaço de projeto.

Finalmente, apesar do framework estar implementado com uma biblioteca reduzida de gabaritos e IPs, a construção de novos gabaritos mais diversificados só poderão ser apresentados após a construção de tais elementos (IPs), evidenciando-se inúmeros cenários envolvendo situações de diversos níveis de complexidade, entretanto os resultados iniciais apresentados nesse projeto estimulam a sua continuidade e propiciam um ambiente de pesquisa na área de sistemas embarcados reconfiguráveis, tão carentes no Brasil.

6.3 Trabalhos Futuros

Durante a construção deste trabalho muitas propostas foram sugeridas para que o framework ADAS-Vision alcançasse uma melhor performance e excedesse algumas de suas limitações. Os trabalhos futuros devem ser direcionados para a implementação de tais propostas, neste sentido se faz necessário: Realizar a implementação por hardware de técnicas de aprendizado de redes neurais, visando reconhecimento e classificação de objetos; Importação e exportação de dados para outras plataformas (MatLab, R, Octave); Integração com OpenCL e OpenGL; e finalmente, a expansão do sistema para uso em outras aplicações de sistemas embarcados tais como o desenvolvimento de VANTS. Porém, tão importante quanto aumentar o número de gabaritos e de IPs é a extração de resultados mais detalhados dos projetos implementados nas placas de FPGA, visando a construção de curvas de pareto-ótima para uma melhor e mais versátil exploração de espaço de projeto. Esta etapa se torna imprescindível para o sucesso desta ferramenta.

Em complemento, o site <<http://www.leandromartinez.com.br/adas-vision>> contém maiores informações sobre este projeto, resultados e documentações deste *framework*.

REFERÊNCIAS

ALTERA. **Nios II Classic Processor Reference Guide**. 2016. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf>. Citado na página 81.

_____. **Nios II Custom Instruction User Guide**. 2016. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_nios2_custom_instruction.pdf>. Citado na página 82.

AMMAR, M.; BAKLOUTI, M.; ABID, M. The performance-energy tradeoff in embedded systems design: A survey of existing design space exploration tools and trends. **International Journal of Computer Science and Information Security**, LJS Publishing, v. 14, n. 5, p. 381, 2016. Citado nas páginas 48 e 59.

BACON, D.; RABBAH, R.; SHUKLA, S. FPGA programming for the masses. **Queue**, ACM, New York, NY, USA, v. 11, n. 2, p. 40:40–40:52, fev. 2013. ISSN 1542-7730. Disponível em: <<http://doi.acm.org/10.1145/2436696.2443836>>. Citado nas páginas 26 e 49.

BAILEY, D. G. **Design for embedded image processing on FPGAs**. [S.l.]: John Wiley & Sons, 2011. Citado nas páginas 33, 43 e 44.

BALARIN, F.; WATANABE, Y.; HSIEH, H.; LAVAGNO, L.; PASSERONE, C.; SANGIOVANNI-VINCENTELLI, A. Metropolis: An integrated electronic system design environment. **Computer**, IEEE, v. 36, n. 4, p. 45–52, 2003. Citado na página 54.

BENGLER, K.; DIETMAYER, K.; FARBER, B.; MAURER, M.; STILLER, C.; WINNER, H. Three decades of driver assistance systems: Review and future perspectives. **IEEE Intelligent Transportation Systems Magazine**, Institute of Electrical and Electronics Engineers (IEEE), v. 6, n. 4, p. 6–22, 2014. Disponível em: <<http://dx.doi.org/10.1109/MITS.2014.2336271>>. Citado na página 38.

BI, Y.; LI, C.; YANG, F. Very high level synthesis for image processing applications. In: **Proceedings of the 10th International Conference on Distributed Smart Camera - ICDSC '16**. Association for Computing Machinery (ACM), 2016. Disponível em: <<http://dx.doi.org/10.1145/2967413.2967414>>. Citado na página 43.

BLEULER, S.; LAUMANN, M.; THIELE, L.; ZITZLER, E. Pisa—a platform and programming language independent interface for search algorithms. In: SPRINGER. **International Conference on Evolutionary Multi-Criterion Optimization**. [S.l.], 2003. p. 494–508. Citado na página 55.

BLUESPEC. **BSV Documentation**. 2017. Disponível em: <<http://wiki.bluespec.com/Home/BSV-Documentation>>. Acesso em: 01/06/2017. Citado nas páginas 41 e 42.

BOBDA, C. Reconfigurable architectures. In: **Introduction to Reconfigurable Computing**. Springer, 2007. ISBN 978-1-4020-6088-5. Disponível em: <http://dx.doi.org/10.1007/978-1-4020-6100-4_2>. Citado na página 31.

- BOEING, A.; BRAUNL, T. Improvcv: Open component based automotive vision. In: **2008 IEEE Intelligent Vehicles Symposium**. [S.l.: s.n.], 2008. Citado nas páginas 53 e 54.
- CALTECH.EDU. **Caltech Pedestrian Detection Benchmark**. 2017. Disponível em: <http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/>. Acesso em: 01/06/2017. Citado na página 40.
- CAMPBELL, K.; HE, L.; YANG, L.; GURUMANI, S.; RUPNOW, K.; CHEN, D. Debugging and verifying SoC designs through effective cross-layer hardware-software co-simulation. In: **Proceedings of the 53rd Annual Design Automation Conference on - DAC '16**. Association for Computing Machinery (ACM), 2016. Disponível em: <<http://dx.doi.org/10.1145/2897937.2898002>>. Citado na página 36.
- CARDOSO, J.; HÜBNER, M. **Reconfigurable Computing: From FPGAs to Hardware/-Software Codesign**. [S.l.]: Springer Science & Business Media, 2011. Citado na página 32.
- COMEMSO. **Comenso ATDF website**. 2017. Disponível em: <<https://www.elektrobit.com/products/eb-assist/adtf/>>. Acesso em: Junho/2017. Citado na página 51.
- COMPTON, K.; HAUCK, S. Reconfigurable computing: a survey of systems and software. **ACM Computing Surveys (csuR)**, ACM, v. 34, n. 2, p. 171–210, 2002. Citado na página 32.
- COUSSY, P.; GAJSKI, D. D.; MEREDITH, M.; TAKACH, A. An introduction to high-level synthesis. **IEEE Design Test of Computers**, v. 26, n. 4, p. 8–17, jul. 2009. ISSN 0740-7475. Citado na página 33.
- COUSSY, P.; MORAWIEC, A. **High-Level Synthesis**. Springer Nature, 2008. Disponível em: <<http://dx.doi.org/10.1007/978-1-4020-8588-8>>. Citado na página 33.
- DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: **In CVPR**. [S.l.: s.n.], 2005. p. 886–893. Citado na página 99.
- DATASUS. **Vítimas de acidentes de trânsito: Base de dados datasus**. 2015. Disponível em: <http://www.vias-seguras.com/os_acidentes/estatisticas/estatisticas_nacionais>. Acesso em: 2016-12-09. Citado nas páginas 26 e 27.
- DAVE, N. H. **Designing a processor in Bluespec**. Dissertação (Mestrado), 2005. Disponível em: <<http://dspace.mit.edu/handle/1721.1/30174>>. Citado na página 41.
- DENSMORE, D.; PASSERONE, R. A platform-based taxonomy for esl design. **IEEE Design Test of Computers**, v. 23, n. 5, p. 359–374, maio 2006. ISSN 0740-7475. Citado na página 33.
- DOLLAR, P.; WOJEK, C.; SCHIELE, B.; PERONA, P. Pedestrian detection: An evaluation of the state of the art. v. 34, n. 4, p. 743–761, abr. 2012. ISSN 0162-8828. Citado nas páginas 89, 90 e 97.
- DÖMER, R.; GERSTLAUER, A.; PENG, J.; SHIN, D.; CAI, L.; YU, H.; ABDI, S.; GAJSKI, D. D. System-on-chip environment: A specc-based framework for heterogeneous mpsoc design. **EURASIP Journal on Embedded Systems**, Hindawi Publishing Corp., v. 2008, p. 5, 2008. Citado na página 57.

DPRF. **Estatísticas de acidentes de trânsito**. 2011. Versão preliminar. Disponível em: <<http://www.dnit.gov.br/rodovias/operacoes-rodoviaras/estatisticas-de-acidentes>>. Citado na página 26.

DRECHSLER, R. Towards formal verification on the system level. In: **Proc. 15th IEEE Int. Workshop Rapid System Prototyping**. [S.l.: s.n.], 2004. p. 2–5. ISSN 1074-6005. Citado na página 34.

GAVRILA, D. M.; MUNDER, S. Multi-cue pedestrian detection and tracking from a moving vehicle. **International Journal of Computer Vision**, Springer Nature, v. 73, n. 1, p. 41–59, jul 2006. Disponível em: <<http://dx.doi.org/10.1007/s11263-006-9038-7>>. Citado na página 40.

GERONIMO, D.; LOPEZ, A. M.; SAPP, A. D.; GRAF, T. Survey of pedestrian detection for advanced driver assistance systems. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Institute of Electrical and Electronics Engineers (IEEE), v. 32, n. 7, p. 1239–1258, jul 2010. Disponível em: <<http://dx.doi.org/10.1109/TPAMI.2009.122>>. Citado na página 38.

GROUP, D. **Dini Group website**. 2017. Disponível em: <http://www.dinigroup.com/product/common/DINI_selection_guide_v540.pdf>. Acesso em: Junho/2017. Citado na página 26.

HA, S.; KIM, S.; LEE, C.; YI, Y.; KWON, S.; JOO, Y.-P. Peace: A hardware-software codesign environment for multimedia embedded systems. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, ACM, v. 12, n. 3, p. 24, 2007. Citado na página 58.

HARALICK, R. M.; SHAPIRO, L. G. Glossary of computer vision terms. **Pattern Recognition**, Elsevier BV, v. 24, n. 1, p. 69–93, jan 1991. Disponível em: <[http://dx.doi.org/10.1016/0031-3203\(91\)90117-N](http://dx.doi.org/10.1016/0031-3203(91)90117-N)>. Citado na página 43.

HEISH, T.-H.; LIN, R.-B. Via-configurable structured asic implementation of openrisc 1200 based soc platform. In: IEEE. **Next-Generation Electronics (ISNE), 2013 IEEE International Symposium on**. [S.l.], 2013. p. 21–24. Citado nas páginas 28 e 32.

ICARUS. **ICARUS Verilog website**. 2017. Disponível em: <<http://iverilog.icarus.com/>>. Acesso em: Junho/2017. Citado na página 73.

IEEE (Ed.). **Verilog Register Transfer Level Synthesis**. Institute of Electrical and Electronics Engineers (IEEE), 2005. Disponível em: <<http://dx.doi.org/10.1109/IEEESTD.2005.339572>>. Citado na página 40.

INTEMPORA. **Intempora editor of RTMaps**. 2017. Disponível em: <<https://intempora.com/products/rmaps.html>>. Citado nas páginas 50 e 52.

ITRS. **System Drivers**. 2011. 2011 - Chapters - SysDrivers.pdf. Disponível em: <<http://www.itrs2.net/2011-its.html>>. Citado nas páginas 26 e 32.

KANGAS, T.; KUKKALA, P.; ORSILA, H.; SALMINEN, E.; HÄNNIKÄINEN, M.; HÄMÄLÄINEN, T. D.; RIIHIMÄKI, J.; KUUSILINNA, K. Uml-based multiprocessor soc design framework. **ACM Transactions on Embedded Computing Systems (TECS)**, ACM, v. 5, n. 2, p. 281–320, 2006. Citado na página 58.

KEINERT, J.; SCHLICHTER, T.; FALK, J.; GLADIGAU, J.; HAUBELT, C.; TEICH, J.; MEREDITH, M. *et al.* Systemcodesigner—an automatic esl synthesis approach by design space exploration and behavioral synthesis for streaming applications. **ACM Transactions on Design**

Automation of Electronic Systems (TODAES), ACM, v. 14, n. 1, p. 1, 2009. Citado na página 57.

KIM, J.; SHIN, H. (Ed.). **Algorithm & SoC Design for Automotive Vision Systems**. [S.l.]: Springer Netherlands, 2014. Citado na página 36.

KIRISCHIAN, V.; GEURKOV, V.; KIRISCHIAN, L. Cost effective reconfigurable architecture for stream processing applications. In: **Proc. Canadian Conf. Electrical and Computer Engineering**. [S.l.: s.n.], 2008. p. 000541–000546. ISSN 0840-7789. Citado na página 32.

KISAČANIN, B.; GELAUTZ, M. **Advances in Embedded Computer Vision**. Springer International Publishing, 2014. Disponível em: <<http://dx.doi.org/10.1007/978-3-319-09387-1>>. Citado na página 38.

LABVIEW. **LabView website**. 2017. Disponível em: <<http://www.ni.com/labview>>. Acesso em: 01/06/2017. Citado nas páginas 52 e 53.

LIEVERSE, P.; WOLF, P. V. D.; VISSERS, K.; DEPRETTERE, E. A methodology for architecture exploration of heterogeneous signal processing systems. **Journal of VLSI signal processing systems for signal, image and video technology**, Springer, v. 29, n. 3, p. 197–207, 2001. Citado na página 56.

MANTYNEVA, J. R. **Automated Design Space Exploration of Transport Triggered Architectures**. Dissertação (Mestrado), 2009. Citado nas páginas 93 e 94.

MARTINEZ, L. A. **Projeto de um sistema embarcado de predição de colisão a pedestres baseado em computação reconfigurável**. Dissertação (Mestrado), 2011. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-27022012-110356/pt-br.php>>. Citado nas páginas 39 e 89.

MATLAB. **MatLab HDL Coder website**. 2017. Disponível em: <<https://www.mathworks.com/products/hdl-coder.html>>. Acesso em: Junho/2017. Citado nas páginas 53, 55, 56 e 83.

MIHAL, A.; KULKARNI, C.; MOSKEWICZ, M.; TSAI, M.; SHAH, N.; WEBER, S.; JIN, Y.; KEUTZER, K.; VISSERS, K.; SAUER, C. *et al.* Developing architectural platforms: A disciplined approach. **IEEE Design & Test of Computers**, IEEE, v. 19, n. 6, p. 6–16, 2002. Citado nas páginas 55 e 56.

NIKOLOV, H.; THOMPSON, M.; STEFANOV, T.; PIMENTEL, A.; POLSTRA, S.; BOSE, R.; ZISSULESCU, C.; DEPRETTERE, E. Daedalus: toward composable multimedia mp-soc design. In: ACM. **Proceedings of the 45th annual Design Automation Conference**. [S.l.], 2008. p. 574–579. Citado na página 57.

NIOSII. **Limitações do NIOS II**. 2017. Disponível em: <<https://www.altera.com/support/support-resources/knowledge-base/solutions/spr361482.html>>. Acesso em: Junho/2017. Citado na página 87.

OLIVEIRA, B. R. do N. **A quality model for critical embedded systems**. Dissertação (Mestrado), 2017. Citado nas páginas 93 e 95.

OLIVEIRA, L. B. R.; GUESSI, M.; FEITOSA, D.; MANTEUFFEL, C.; GALSTER, M.; OQUENDO, F.; NAKAGAWA, E. Y. An investigation on quality models and quality attributes for embedded systems. 2013. Citado na página 93.

OLIVEIRA, M. F. da S. **Model driven engineering methodology for design space exploration of embedded systems**. Tese (Doutorado), 2013. Disponível em: <<http://hdl.handle.net/10183/102694>>. Citado na página 54.

OPENCV. **OpOpen website**. 2017. Disponível em: <<http://opencv.org/>>. Acesso em: Junho/2017. Citado na página 83.

PATEL, S. K. S. H. D. **Ingredients for Successful System Level Automation Design Methodology**. [S.l.]: Springer-Verlag GmbH, 2008. ISBN 1402084714. Citado na página 33.

PIMENTEL, A. D.; HERTZBETGER, L.; LIEVERSE, P.; WOLF, P. V. D.; DEPRETTERE, E. Exploring embedded-systems architectures with artemis. **Computer**, IEEE, v. 34, n. 11, p. 57–63, 2001. Citado na página 57.

PU, J.; BELL, S.; YANG, X.; SETTER, J.; RICHARDSON, S.; RAGAN-KELLEY, J.; HOROWITZ, M. **Programming Heterogeneous Systems from an Image Processing DSL**. 2016. Citado na página 49.

QT. **QT Creator**. 2017. Disponível em: <<https://www.qt.io/>>. Acesso em: Junho/2017. Citado nas páginas 63 e 67.

RETTKOWSKI, J.; BOUTROS, A.; GÖHRINGER, D. Hw/sw co-design of the {HOG} algorithm on a xilinx zynq soc. **Journal of Parallel and Distributed Computing**, p. –, 2017. ISSN 0743-7315. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0743731517301569>>. Citado na página 101.

REYES, V.; KRUIJTZER, W.; BAUTISTA, T.; ALKADI, G.; NÚÑEZ, A. A unified system-level modeling and simulation environment for mpsoe design: Mpeg-4 decoder case study. In: EUROPEAN DESIGN AND AUTOMATION ASSOCIATION. **Proceedings of the conference on Design, automation and test in Europe: Proceedings**. [S.l.], 2006. p. 474–479. Citado na página 58.

SCHUBERT, R.; RICHTER, E.; MATTERN, N.; LOBEL, H. Rapid prototyping of adas and its applications on the example of a vision-based vehicle tracking system. 2012. Citado nas páginas 50 e 51.

SEBESTA, R. W. **Concepts of Programming Languages**. 10. ed. [S.l.: s.n.], 2012. Citado na página 31.

SHUKLA, S. K.; PIXLEY, C.; SMITH, G. Guest editors' introduction: The true state of the art of esl design. **IEEE Design Test of Computers**, v. 23, n. 5, p. 335–337, maio 2006. ISSN 0740-7475. Citado na página 33.

SUTHERLAND, S.; DAVIDMANN, S.; FLAKE, P. **SystemVerilog for Design**. [S.l.]: Springer-Verlag GmbH, 2006. ISBN 0387333991. Citado na página 40.

TECHMER, A. Application development of camera-based driver assistance systems on a programmable multi-processor architecture. In: **Proc. IEEE Intelligent Vehicles Symp.** [S.l.: s.n.], 2007. p. 1211–1216. ISSN 1931-0587. Citado nas páginas 33 e 38.

TEICH, J. Hardware/software codesign: The past, the present, and predicting the future. **Proceedings of the IEEE**, Institute of Electrical and Electronics Engineers (IEEE), v. 100, n. Special Centennial Issue, p. 1411–1430, may 2012. Disponível em: <<http://dx.doi.org/10.1109/JPROC.2011.2182009>>. Citado nas páginas 34 e 35.

TERASIC. **FPGA Boards**. 2016. Disponível em: <<http://www.terasic.com.tw/>>. Citado na página 62.

TESSIER, R.; POCEK, K.; DEHON, A. Reconfigurable computing architectures. **Proceedings of the IEEE**, Institute of Electrical and Electronics Engineers (IEEE), v. 103, n. 3, p. 332–354, mar 2015. Disponível em: <<http://dx.doi.org/10.1109/JPROC.2014.2386883>>. Citado na página 34.

WANG, P. H.; COLLINS, J. D.; WEAVER, C. T.; KUTTANNA, B.; SALAMIAN, S.; CHINYA, G. N.; SCHUCHMAN, E.; SCHILLING, O.; DOIL, T.; STEIBL, S.; WANG, H. Intel®atom™ processor core made FPGA-synthesizable. In: **Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays**. New York, NY, USA: ACM, 2009. (FPGA '09), p. 209–218. ISBN 978-1-60558-410-2. Disponível em: <<http://doi.acm.org/10.1145/1508128.1508160>>. Citado na página 32.

WANG, Q.; PANG, J.; QIN, L.; JIANG, S.; HUANG, Q. Justifying the importance of color cues in object detection: A case study on pedestrian. In: _____. **The Era of Interactive Media**. New York, NY: Springer New York, 2013. p. 387–397. ISBN 978-1-4614-3501-3. Disponível em: <http://dx.doi.org/10.1007/978-1-4614-3501-3_32>. Citado na página 99.

YAMAURA, M.; ARECHIGA, N.; SHIRAISHI, S.; EISELE, S.; HITE, J.; NEEMA, S.; SCOTT, J.; BAPTY, T. **ADAS Virtual Prototyping using Modelica and Unity Co-simulation via OpenMETA**. 2016. Disponível em: <<http://www.isis.vanderbilt.edu/node/4767>>. Citado na página 36.

YOSHIDA, J. **Tesla's Fatal Crash: 6 Unanswered Questions**. 2016. Disponível em: <http://www.eetimes.com/document.asp?doc_id=1330060>. Acesso em: Junho/2017. Citado na página 26.

ZHANG, Z.; PORTER, J.; EYISI, E.; KARSAI, G.; KOUTSOUKOS, X.; SZTIPANOVITS, J. Co-simulation framework for design of time-triggered cyber physical systems. In: **Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems**. New York, NY, USA: ACM, 2013. (ICCPs '13), p. 119–128. ISBN 978-1-4503-1996-6. Disponível em: <<http://doi.acm.org/10.1145/2502524.2502541>>. Citado na página 35.

ZHAO, M. techreport, **Advanced Driver Assistant System: Threats, requirements, security solutions**. 2015. Disponível em: <<http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/advanced-driver-assistant-system-paper.pdf>>. Citado na página 37.

DOCUMENTAÇÃO DOS BLOCOS DE HARDWARE

A.1 Exemplo Código Fonte de um IP

Código-fonte 3 – Subrotina de um bloco IP compilado para Verilog

```
1:
2: `ifdef BSV_ASSIGNMENT_DELAY
3: `else
4: `define BSV_ASSIGNMENT_DELAY
5: `endif
6:
7: module mkCalcular(CLK,
8:     RST_N,
9:     put_r,
10:    put_g,
11:    put_b,
12:    EN_put,
13:    RDY_put,
14:
15:    get,
16:    RDY_get);
17: input  CLK;
18: input  RST_N;
19:
20: // action method put
21: input  [31 : 0] put_r;
22: input  [31 : 0] put_g;
```

```

23:  input  [31 : 0] put_b;
24:  input  EN_put;
25:  output RDY_put;
26:
27:  // value method get
28:  output [7 : 0] get;
29:  output RDY_get;
30:
31:  // signals for module outputs
32:  wire [7 : 0] get;
33:  wire RDY_get, RDY_put;
34:
35:  // register value
36:  reg [31 : 0] value;
37:  wire [31 : 0] value\$_D_IN;
38:  wire value\$_EN;
39:
40:  // rule scheduling signals
41:  wire CAN_FIRE_put, WILL_FIRE_put;
42:
43:  // remaining internal signals
44:  wire [31 : 0]
      IF_put_r_PLUS_put_g_PLUS_put_b_BIT_31_THEN_NEG_ETC___d11,
45:  put_r_PLUS_put_g_PLUS_put_b___d10,
46:  x__h131;
47:
48:  // action method put
49:  assign RDY_put = 1'd1 ;
50:  assign CAN_FIRE_put = 1'd1 ;
51:  assign WILL_FIRE_put = EN_put ;
52:
53:  // value method get
54:  assign get = value[7:0] ;
55:  assign RDY_get = 1'd1 ;
56:
57:  // register value
58:  assign value\$_D_IN =
59:      put_r_PLUS_put_g_PLUS_put_b___d10[31] ?
60:      -
      IF_put_r_PLUS_put_g_PLUS_put_b_BIT_31_THEN_NEG_ETC___d11 :
61:      IF_put_r_PLUS_put_g_PLUS_put_b_BIT_31_THEN_NEG_ETC___d11 ;

```

```

62:  assign value\ $EN = EN_put ;
63:
64:  // remaining internal signals
65:  assign
        IF_put_r_PLUS_put_g_PLUS_put_b_BIT_31_THEN_NEG_ETC___d11 =
66:      x__h131 / 32'd3 ;
67:  assign put_r_PLUS_put_g_PLUS_put_b___d10 = put_r + put_g +
        put_b ;
68:  assign x__h131 =
69:      put_r_PLUS_put_g_PLUS_put_b___d10[31] ?
70:      -put_r_PLUS_put_g_PLUS_put_b___d10 :
71:      put_r_PLUS_put_g_PLUS_put_b___d10 ;
72:
73:  // handling of inlined registers
74:
75:  always@(posedge CLK)
76:  begin
77:      if (value\ $EN) value <= 'BSV_ASSIGNMENT_DELAY value\ $D_IN;
78:  end
79:
80:  // synopsys translate_off
81:  'ifdef BSV_NO_INITIAL_BLOCKS
82:  'else // not BSV_NO_INITIAL_BLOCKS
83:  initial
84:  begin
85:      value = 32'hAAAAAAAA;
86:  end
87:  'endif // BSV_NO_INITIAL_BLOCKS
88:  // synopsys translate_on
89: endmodule // mkCalcular

```

Exemplo de Código: IP TestBench Verilog

Código-fonte 4 – Subrotina de teste de bancada compilado em verilog

```

1:
2: 'ifdef BSV_ASSIGNMENT_DELAY
3: 'else
4: 'define BSV_ASSIGNMENT_DELAY
5: 'endif
6:
7: module rgbConvert(CLK,

```

```

8:      RST_N);
9:  input  CLK;
10:  input  RST_N;
11:
12:  integer file, fileSize, count, r, xCount;
13:  reg [31 : 0] xOut;
14:
15:  // register arq
16:  reg [31 : 0] arq;
17:  wire [31 : 0] arq\$_D_IN;
18:  wire arq\$_EN;
19:
20:  // register arqOut
21:  reg [31 : 0] arqOut;
22:  wire [31 : 0] arqOut\$_D_IN;
23:  wire arqOut\$_EN;
24:
25:  // register cnt
26:  reg [2 : 0] cnt;
27:  wire [2 : 0] cnt\$_D_IN;
28:  wire cnt\$_EN;
29:
30:  // ports of submodule p
31:  wire [31 : 0] p\$_put_b, p\$_put_g, p\$_put_r;
32:  wire [7 : 0] p\$_get;
33:  wire p\$_EN_put;
34:
35:  // rule scheduling signals
36:  wire CAN_FIRE_RL_executar,
37:        CAN_FIRE_RL_fim,
38:        CAN_FIRE_RL_openFiles,
39:        CAN_FIRE_RL_saveData,
40:        WILL_FIRE_RL_executar,
41:        WILL_FIRE_RL_fim,
42:        WILL_FIRE_RL_openFiles,
43:        WILL_FIRE_RL_saveData;
44:
45:  // inputs to muxes for submodule ports
46:  wire [2 : 0] MUX_cnt\$_write_1__VAL_1;
47:
48:  // remaining internal signals
49:  reg [31 : 0] TASK_fopen___d23, TASK_fopen___d24, b__h624,

```

```

        b__h691, b__h759;
50:
51:  // submodule p
52:  mkCalcular p(.CLK(CLK),
53:              .RST_N(RST_N),
54:              .put_b(p\$_put_b),
55:              .put_g(p\$_put_g),
56:              .put_r(p\$_put_r),
57:              .EN_put(p\$_EN_put),
58:              .RDY_put(),
59:              .get(p\$_get),
60:              .RDY_get());
61:
62:  // rule RL_fim
63:  assign CAN_FIRE_RL_fim = cnt == 3'd3 ;
64:  assign WILL_FIRE_RL_fim = CAN_FIRE_RL_fim ;
65:
66:  // rule RL_saveData
67:  assign CAN_FIRE_RL_saveData = cnt == 3'd2 ;
68:  assign WILL_FIRE_RL_saveData = CAN_FIRE_RL_saveData ;
69:
70:  // rule RL_executar
71:  assign CAN_FIRE_RL_executar = cnt == 3'd1 ;
72:  assign WILL_FIRE_RL_executar = CAN_FIRE_RL_executar ;
73:
74:  // rule RL_openFiles
75:  assign CAN_FIRE_RL_openFiles = cnt == 3'd0 ;
76:  assign WILL_FIRE_RL_openFiles = CAN_FIRE_RL_openFiles ;
77:
78:  // inputs to muxes for submodule ports
79:  assign MUX_cnt\$_write_1__VAL_1 = (fileSize <= count) ? 3'd3 :
        3'd2 ;
80:
81:  // register arq
82:  assign arq\$_D_IN = TASK_fopen__d23 ;
83:  assign arq\$_EN = CAN_FIRE_RL_openFiles ;
84:
85:  // register arqOut
86:  assign arqOut\$_D_IN = TASK_fopen__d24 ;
87:  assign arqOut\$_EN = CAN_FIRE_RL_openFiles ;
88:
89:  // register cnt

```

```

90:   assign cnt\$_D_IN = WILL_FIRE_RL_executar ? MUX_cnt\
      $write_1__VAL_1 : 3'd1 ;
91:   assign cnt\$_EN =
92:       WILL_FIRE_RL_executar || WILL_FIRE_RL_openFiles ||
93:       WILL_FIRE_RL_saveData ;
94:
95:   // submodule p
96:   assign p\$_EN_put = WILL_FIRE_RL_executar && fileSize > count
      ;
97:   assign p\$_put_r = b__h624 ;
98:   assign p\$_put_g = b__h691 ;
99:   assign p\$_put_b = b__h759 ;
100:
101:   // handling of inlined registers
102:
103:   always@(posedge CLK)
104:   begin
105:       if (!RST_N)
106:       begin
107:           cnt <= 'BSV_ASSIGNMENT_DELAY 3'd0;
108:       end
109:       else
110:       begin
111:           if (cnt\$_EN) cnt <= 'BSV_ASSIGNMENT_DELAY cnt\$_D_IN;
112:       end
113:       if (arq\$_EN) arq <= 'BSV_ASSIGNMENT_DELAY arq\$_D_IN;
114:       if (arqOut\$_EN) arqOut <= 'BSV_ASSIGNMENT_DELAY arqOut\
      $_D_IN;
115:   end
116:
117:   // synopsys translate_off
118:   `ifdef BSV_NO_INITIAL_BLOCKS
119:   `else // not BSV_NO_INITIAL_BLOCKS
120:   initial
121:   begin
122:       arq = 32'hAAAAAAAA;
123:       arqOut = 32'hAAAAAAAA;
124:       cnt = 3'h2;
125:       count = 0;
126:       fileSize = 0;
127:       xCount = 0;
128:   end

```

```
129:  'endif // BSV_NO_INITIAL_BLOCKS
130:  // synopsys translate_on
131:
132:  // handling of system tasks
133:
134:  // synopsys translate_off
135:  always@(negedge CLK)
136:  begin
137:      #0;
138:      if (RST_N) if (WILL_FIRE_RL_fim) \ $fclose(arq);
139:      if (RST_N) if (WILL_FIRE_RL_fim)
140:      begin
141:          \ $fclose(arqOut);
142:      end
143:      if (RST_N) if (WILL_FIRE_RL_fim) \ $display("Concluido.");
144:      if (RST_N) if (WILL_FIRE_RL_fim) \ $finish(32'd0);
145:      if (RST_N)
146:      if (WILL_FIRE_RL_saveData)
147:      begin
148:          if (xCount == 0) xOut[7:0] = p\ $get;
149:          if (xCount == 1) xOut[15:8] = p\ $get;
150:          if (xCount == 2) xOut[23:16] = p\ $get;
151:          if (xCount == 3)
152:          begin
153:              xOut[31:24] = p\ $get;
154:              \ $fwriteb(TASK_fopen__d24, "\%U", xOut[31:0]);
155:              xCount = 0;
156:          end
157:          else
158:              xCount = xCount + 1;
159:      end
160:
161:      if (RST_N)
162:      if (WILL_FIRE_RL_executar)
163:      begin
164:          r = \ $fseek(TASK_fopen__d23, count, 0);
165:          r = \ $fscanf(TASK_fopen__d23, "\%c\%c\%c", b__h624,
166:              b__h691, b__h759);
167:          count = count + 3;
168:          #0;
169:      end
```

```

170:     if (RST_N)
171:         if (WILL_FIRE_RL_openFiles)
172:     begin
173:         TASK_fopen___d23 = \fopen("in.dat", "r");
174:         #0;
175:         r = \fseek(TASK_fopen___d23, 0, 2);
176:         fileSize = \ftell(TASK_fopen___d23);
177:     end
178:
179:     if (RST_N)
180:         if (WILL_FIRE_RL_openFiles)
181:     begin
182:         TASK_fopen___d24 = \fopen("out.dat", "wb");
183:         #0;
184:     end
185:
186:     if (RST_N) if (WILL_FIRE_RL_openFiles) \display("Inicio.")
        ;
187:
188:     end
189: // synopsys translate_on
190: endmodule // rgbConvert

```

Exemplo de Código: IP Make

Código-fonte 5 – Subrotina de arquivo em lote para compilação pelo simulador

```

1: @echo off
2: CD \%FrameworkDir%\%IP033
3: del IP033.vvp
4: iverilog -g2005-sv -o IP033.vvp -DTOP=rgbConvert -
    DBSV_ASSIGNMENT_DELAY=#1 -s main ../Bluespec/Verilog/main.v
    mkCalcular.v rgbConvert.v
5: @echo Compilado.

```

Exemplo de Código: IP Run VVP

Código-fonte 6 – Subrotina de arquivo em lote para execução da simulação

```

1: @echo off
2: CD \%FrameworkDir%\%IP033
3: del out.dat
4: vvp IP033.vvp

```


A Figura 60 mostra como está estruturado o sistema de diretórios. Ao iniciar um novo projeto, o sistema cria automaticamente as estruturas.

Figura 60 – Formação dos Diretórios do Sistema

Fonte: Elaborada pelo autor.

SITES RELACIONADOS AO PROJETO

A.0.1 Bibliotecas e Softwares utilizados

Para escolha da plataforma de software e das ferramentas associadas foi levado em consideração a continuação futura da pesquisa. Nesse sentido, foram priorizadas ferramentas com licença de uso livre e preferencialmente as de código aberto. Também houve a preocupação em usar tecnologias atualizadas e amplamente difundidas na comunidade científica. A seguir um resumo das ferramentas que serão utilizadas:

Eclipse: IDE (Ambiente Integrado de Desenvolvimento) open source multiplataforma de desenvolvimento de software desenvolvido em Java criado pela IBM e disponibilizando como software livre para a comunidade. O Eclipse é um IDE Java bastante utilizado no mundo e compõe uma das ferramentas de produção do software embarcado da Altera o NIOS II IDE (NIOS II, 2013). Para exibir elementos gráficos faz uso da SWT, acessando as bibliotecas gráficas nativas do sistema operacional. A orientação ao desenvolvimento é baseada em plug-ins onde disponibiliza centenas de aplicações que atendem as diferentes necessidades de programadores, e também com amplo suporte.

GEF: Framework desenvolvido para a plataforma Eclipse usado para criar editores gráficos com suporte a vários diagramas. Esses diagramas oferecem recursos de simples edição para domínios específicos e são adaptados para representar dados graficamente. É normalmente utilizado para criar o código para o modelo de dados (GEF, 2013).

OpenCV: Biblioteca open source multiplataforma desenvolvida nas linguagens de programação C/C++ pela Intel, para o desenvolvimento de aplicativos na área de Visão Computacional. O OpenCV disponibiliza módulos de processamento de imagens e vídeo E/S, estrutura de dados, álgebra linear, GUI (Interface gráfica do utilizador) básica com sistema de janelas, e disponibiliza mais de 350 algoritmos de Visão Computacional como filtros de imagem e reconhecimento de objetos. Sua arquitetura também permite o processamento em tempo

real de imagens com suporte a linguagens: Java, Python e Visual Basic, podendo ser incorporada a biblioteca de aplicativos. (<<http://opencv.org/>>)

GCC: Conjunto de compiladores de linguagens de programação distribuído pela Free Software Foundation (FSF) que permite compilar o código-fonte (software) em binários executáveis para as várias plataformas. Atualmente suporta diversas linguagens como: C/C++, Fortran, Ada, Java e outras (GCC, 2013).

Para escolha da plataforma de hardware e ferramentas associadas, levou-se em consideração a disponibilidade de recursos e do conhecimento acumulado dos laboratórios de pesquisa da universidade. A pesquisa será desenvolvida no Laboratório de Computação Reconfigurável (LCR) do departamento de computação do ICMC-USP onde são utilizadas ferramentas da Altera desde 1995, possuindo know-how que será aproveitado para facilitar a execução do projeto. A seguir um resumo das tecnologias que serão utilizadas:

Quartus II: Ferramenta EDA (Eletronic Design Automation) da Altera para construção de projetos e que oferece um ambiente unificado de desenvolvimento para sistemas baseados em dispositivos lógicos programáveis. O Quartus II visa diminuir o tempo investido no projeto a fim de aumentar a produtividade dos desenvolvedores, permitindo utilizar uma interface gráfica ou uma interface de linha de comando e scripts de automatização. (<<https://www.altera.com/>>)

QSys: Ferramenta integrada ao Quartus II com uma interface para criação de SoPCs compostos por processadores, memórias, portas paralelas e seriais e periféricos definidos pelo usuário. Esta ferramenta combina os componentes automaticamente gerando a lógica de barramentos, interconexões e integração. (<<https://www.altera.com/>>)

Nios II Embedded Design Suite: Ferramenta de desenvolvimento de software para a família de processadores embarcados Nios II. Trata-se de um aplicativo baseado no projeto open source de IDE Eclipse. As tarefas de gerenciamento de projeto, edição de código fonte, compilação e depuração são desempenhadas através desta ferramenta. (<<https://www.altera.com/>>)

Model SIM: Aplicativo voltado para a simulação e depuração de hardware utilizando dispositivos FPGA e ASIC. O software trabalha com as linguagens de descrição de hardware (HDLs) como o Verilog, o VHDL e o SystemC. (<<https://www.altera.com/>>)

Bluespec: Linguagem de descrição e síntese de hardware fortemente tipada, que oferece níveis de abstração com uma semântica padronizada deixando arquitetura facilmente interpretável. O projetista pode ter previsibilidade e o controle da arquitetura com o uso da linguagem, permitindo gerar uma implementação de qualidade e código totalmente sintetizável. (<<http://www.bluespec.com/>>)