UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

EXTRAÇÃO E RECONHECIMENTO DE CARACTERES ÓPTICOS A PARTIR DO CO-PROJETO DE *HARDWARE* E *SOFTWARE* SOBRE PLATAFORMA RECONFIGURÁVEL

DISSERTAÇÃO DE MESTRADO

Gustavo Fernando Dessbesell

Santa Maria, RS, Brasil
2008

EXTRAÇÃO E RECONHECIMENTO DE CARACTERES ÓPTICOS A PARTIR DO CO-PROJETO DE *HARDWARE* E *SOFTWARE* SOBRE PLATAFORMA RECONFIGURÁVEL

por

Gustavo Fernando Dessbesell

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-Graduação em Engenharia Elétrica, Área de Concentração em Processamento de Energia, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Engenharia Elétrica**.

Orientador: Prof. Dr. João Baptista dos Santos Martins

Santa Maria, RS, Brasil

2008

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia Elétrica

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

EXTRAÇÃO E RECONHECIMENTO DE CARACTERES ÓPTICOS A PARTIR DO CO-PROJETO DE *HARDWARE* E *SOFTWARE* SOBRE PLATAFORMA RECONFIGURÁVEL

elaborada por
Gustavo Fernando Dessbesell

como requisito parcial para obtenção do grau de
Mestre em Engenharia Elétrica

COMISSÃO EXAMINADORA:

_____
Prof. Dr. João Baptista dos Santos Martins
(Presidente/Orientador)

_____
Prof. Dr. Rolf Fredi Molz, (UNISC)
(Co-Orientador)

_____
Prof. Dr. Giovani Baratto, (UFSM)

_____
Prof. Dr. Renato Perez Ribas, (UFRGS)

Santa Maria, 7 de março de 2008

Dedicado a todos aqueles que
nunca deixaram de me apoiar.

AGRADECIMENTOS

Ao final desta importante etapa da minha vida, são tantas as pessoas a quem devo agradecimentos, que nem sei por quem começar. Bem, comecemos pelo início, então.

Aos professores João Baptista dos S. Martins e César R. Rodrigues, tanto pela oportunidade que me foi dada no sentido de modestamente contribuir para a consolidação do GMicro, quanto pelos ensinamentos prestados. Igualmente agradeço aos sempre prestativos professores André L. Aita, Alice de J. Kozakevicius, Raul Ceretta Nunes, Giovani Baratto e Alexandre Campos pelos esclarecimentos e contribuições fornecidos. Em especial, agradeço novamente ao professor João Baptista dos S. Martins, pela orientação no trabalho e confiança que sempre depositou em mim.

Ao CNPq e à FINEP pelo apoio financeiro, o qual foi imprescindível para o desenvolvimento e conclusão deste trabalho.

Aos colegas do GMicro, pela hospitaleira recepção, pelo companheirismo, por serem prestativos nos momentos em que precisei de auxílio, enfim, pelo ótimo ambiente de trabalho proporcionado pela convivência com os mesmos. Um agradecimento especial ao colega Márcio A. Pacheco, amigo de longa data.

Ao meu co-orientador, professor Rolf F. Molz, pelas valiosas contribuições fornecidas e o apoio prestado.

Aos bons exemplos de servidores desta universidade, que sempre se esmeraram para fornecer uma resposta rápida e correta às consultas e solicitações que eu lhes fiz, contribuindo assim, mesmo que por vezes indiretamente, na conclusão deste trabalho.

A toda minha família, em especial aos meus pais, Gilberto e Alice, pelos irrestritos apoio, compreensão e incentivo que me foram dados ao longo destes 29 anos.

À minha amada, compreensiva, incentivadora e companheira esposa Fabrícia, que certamente é uma das pessoas a quem esta dissertação é dedicada.

Ao Antônio, pela capacidade de me fazer rir, mesmo depois de um dia “daqueles”.

A todos os que de uma maneira ou outra contribuíram para a conclusão deste trabalho e que porventura eu tenha esquecido de mencionar, o meu muito obrigado.

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Engenharia Elétrica
Universidade Federal de Santa Maria

EXTRAÇÃO E RECONHECIMENTO DE CARACTERES ÓPTICOS A PARTIR DO CO-PROJETO DE *HARDWARE* E *SOFTWARE* SOBRE PLATAFORMA RECONFIGURÁVEL

AUTOR: GUSTAVO FERNANDO DESSBESELL
ORIENTADOR: PROF. DR. JOÃO BAPTISTA DOS SANTOS MARTINS
CO-ORIENTADOR: PROF. DR. ROLF FREDI MOLZ
Data e Local da Defesa: Santa Maria, 7 de março de 2008.

Este trabalho apresenta a implementação e análise de um sistema voltado à extração e reconhecimento de caracteres ópticos a partir do co-projeto de *hardware* e *software* sobre uma plataforma reconfigurável. Por conta da importância atribuída ao sentido da visão, sistemas artificiais capazes de emular as tarefas envolvidas neste processo biológico têm sido alvo de pesquisas desde o surgimento dos primeiros computadores digitais, na década de 60. Tendo em vista a recente evolução experimentada na área da computação configurável, surge uma tendência natural à pesquisa e desenvolvimento de sistemas heterogêneos (compostos por uma combinação de blocos de *hardware* e *software*) de visão artificial baseados em tal plataforma. Dentre os principais benefícios proporcionados por sistemas em *chip* podem ser citados a redução no consumo de potência, custos financeiros e área física. Neste sentido, tomando como estudo de caso um Sistema de Reconhecimento de Placas de Licenciamento Veicular (SRPLV), o foco do trabalho está situado na implementação das etapas de localização e reconhecimento de caracteres, sendo o particionamento dos blocos de *hardware* e *software* baseado em heurísticas de custo-benefício. Inicialmente é realizada a implementação de uma versão totalmente em *software* do sistema aqui proposto, sobre plataforma x86, no intuito de avaliar os diversos métodos passíveis de implementação, bem como o de possibilitar um parâmetro de comparação com a versão embarcada do sistema. Os métodos avaliados dizem respeito à etapa de localização de caracteres, haja vista a definição *à priori* do emprego de Redes Neurais Artificiais no reconhecimento dos mesmos. A partir dos resultados obtidos por esta avaliação é realizada a implementação da versão embarcada do sistema, tendo como plataforma um FPGA. Nesta versão, a etapa de localização de caracteres é implementada como um bloco dedicado de *hardware*, enquanto a de reconhecimento constitui-se num *software* executado sobre um microprocessador fisicamente embutido no interior do FPGA. Considerando uma freqüência de operação 10 vezes superior para o processador da plataforma x86, bem como o fato da maior parte do *hardware* embarcado utilizar um *clock* menor ou igual a 25 MHz, o principal resultado consiste no ganho de 2,25 vezes no tempo de execução obtido na segunda versão do sistema. No tocante à capacidade de reconhecimento de placas, os sistemas são equivalentes, sendo capazes de reconhecê-las corretamente em 51,62% das vezes, no melhor caso. Além de SRPLVs, o sistema aqui desenvolvido pode ser empregado na criação de outras aplicações que envolvam a problemática do reconhecimento de caracteres óticos, como reconhecimento automático de placas de trânsito e do número de série de itens numa linha de produção.

Palavras-chave: visão computacional; co-projeto de *software* e *hardware*; FPGA, redes neurais artificiais, sistemas embarcados.

ABSTRACT

Master Dissertation
Programa de Pós-Graduação em Engenharia Elétrica
Universidade Federal de Santa Maria

EXTRACTION AND RECOGNITION OF OPTICAL CHARACTERS BASED ON HARDWARE AND SOFTWARE CO-DESIGN OVER RECONFIGURABLE PLATFORM

AUTHOR: GUSTAVO FERNANDO DESSBESELL
ADVISER: PROF. DR. JOÃO BAPTISTA DOS SANTOS MARTINS
CO-ADVISER: PROF. DR. ROLF FREDI MOLZ
Date and Place of Examination: Santa Maria, March 7th, 2008.

This work presents the implementation and analysis of a system devoted to the extraction and recognition of optical characters which is based on the hardware and software co-design methodology and built over a reconfigurable platform. Since vision is a very important sense, the research in the field of artificial vision systems has been carried out since the very beginning of the digital era, in the early 60's. Taking into account the recent evolution experienced by the configurable computing area, a new tendency of research and development of heterogeneous artificial vision systems emerges. Among the main benefits provided by the so called systems on chip are the reduction of power dissipation, financial costs and physical area. In this sense, taking a License Plate Recognition System (LPRS) as a case study, the focus of this work is the implementation of the character localization and recognition steps, while the partitioning of hardware and software resources is based in cost-benefit heuristics. Initially, a software-only version of the system is build over an x86 platform. More than to allow the evaluation of several character localization related methods, this software-only version is also intended to be used as parameter of comparison for the embedded version of the system. Regarding the character recognition step, it is performed by the means of an Artificial Neural Network. Based on the results provided by the software-only evaluation system, the implementation of the embedded version is performed, considering an FPGA as platform. In this embedded version, the character localization step consists of a dedicated hardware block, while the character recognition step comprises a piece of software executed in a microprocessor that is physically implemented inside the FPGA. Taking into account a 10 times higher frequency of operation for the processor of the x86 platform, as well as the fact that most of the embedded hardware block employs a clock frequency smaller or equal to 25 MHz, the most noticeable result is the 2.25 times faster speed of processing achieved by the embedded version. Regarding the plate recognition capability, both systems have the same performance, being able to successfully recognize plates in 51.62 % of the cases (considering the best case). Beyond LPRSs, the system developed here could also be employed to build other applications that require optical character recognition features, such as automatic traffic signs recognition and serial number reading of items in a production line.

Keywords: *computer vision*; *hardware and software co-design*; FPGA, *artificial neural networks, embedded systems*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico para Digital
ANPR	*Automatic Number Plate Recognition*
ASCII	*American Standard Code for Information Interchange*
ASIC	*Application Specific Integrated Circuit*
AVI	*Automatic Vehicle Identification*
BS	*Barrel Shifter*
BSB	*Base System Builder*
BSP	*Board Support Package*
CAD	*Computer Aided Design*
CD	*Compact Disc*
CISC	*Complex Instruction Set Computer*
CMYK	*Cian, Magenta, Yellow, Key (black)*
CPR	*Car Plate Recognition*
CPR	*Car Plate Reader*
D/A	Digital para Analógico
DCM	*Digital Clock Manager*
DDR	*Double Data Rate*
DSP	*Digital Signal Processor/Processing*
DVD	*Digital Vídeo Disc*
E/S	Entrada/Saída
EDK	*Embedded Development Kit*
EQM	Erro Quadrático Médio
FPGA	*Field Programmable Gate Array*
GPS	*Global Positioning System*
HDL	*Hardware Description Language*
IPIC	*Intellectual Property Interconnect*
IPIF	*Intellectual Property Interface*
ISE	*Integrated Software Environment*
LCD	*Liquid Crystal Display*
LED	*Light Emission Diode*
LMS	*Least Mean Square*
LUT	*Look Up Table*
MP3	*MPEG-1 Audio Layer 3*
OCR	*Optical Character Recognition*
OPB	*On-chip Peripheral Bus*
PC	*Personal Computer*
PDA	*Personal Digital Assistant*
PET	*Positron Emission Tomography*
PLB	*Processor Local Bus*

PMC	*Perceptron* de Múltiplas Camadas
PP	Parada Prematura
RAM	*Random Access Memory*
RBF	*Radial Basis Function*
RGB	*Red, Green, Blue*
RISC	*Reduced Instruction Set Computer*
RNA	Rede Neural Artificial
ROM	*Read-Only Memory*
RPLV	Reconhecimento de Placas de Licenciamento Veicular
RS-232	*Recommended Standard*-232
RTOS	*Real-Time Operating System*
SDK	*Software Development Kit*
SDRAM	*Synchronous Dynamic Random Access Memory*
SO	Sistema Operacional
SoC	*System on Chip*
SRPLV	Sistema de Reconhecimento de Placas de Licenciamento Veicular
UART	*Universal Asynchronous Receiver-Transmitter*
ULA	Unidade Lógica e Aritmética
UPF	Unidade de Ponto Flutuante
XPS	*Xilinx Platform Studio*
XST	*Xilinx Synthesis Tool*

LISTA DE SÍMBOLOS

L	Número de níveis de cinza de uma imagem
N	Número total de *pixels* numa imagem/janela
n_i	Número de *pixels* que possuem o tom de cinza *i*
x_i	Tom de cinza do do *pixel i*
t^*	Valor ótimo para limiar de binarização
σ_B^2	Variância inter-classes
σ_T^2	Variância total
ω_0	Histograma Cumulativo
ω_1	Complemento de ω_0
μ_T	Área Total de Intensidade
μ_t	Área Cumulativa de Intensidade
$\bar{x}$	Média dos tons dos *pixels* contidos numa janela de processamento
σ	Desvio padrão dos tons dos *pixels* contidos numa janela de processamento
$N_4(p)$	Os 4 vizinho de *p*
$N_D(p)$	Os 4 vizinhos diagonais de *p*
$N_8(p)$	Os 8 vizinhos de *p* ($N_4(p) \cup N_8(p)$)
$y_i(n)$	Valor atribuído ao neurônio *i* (camada entrada) na iteração *n*
$w_{ji}(n)$	Peso da conexão entre os neurônios *j* e *i* na iteração *n*
$\upsilon_j(n)$	Campo local induzido do neurônio *j* (camada escondida) na iteração *n*
$\varphi_j(\bullet)$	Valor do sinal de função do neurônio *j* (camada entrada)
$e_k(n)$	Erro apurado para o neurônio *k* (camada de saída) na iteração *n*
$\xi(n)$	Valor instantâneo da energia de erro para um neurônio da camada de saída na iteração *n*
$\xi_{av}(n)$	Energia de erro quadrática média na iteração *n*
$\Delta w_{kj}(n)$	Valor da parcela de ajuste aplicada à conexão entre os neurônios *k* e *j* na iteração *n*
$\delta_k(n)$	Gradiente local do neurônio *k* na iteração *n*
$\delta_j(n)$	Gradiente local do neurônio *j* na iteração *n*
η	Taxa de aprendizado
α	Termo de momento

SUMÁRIO

1 INTRODUÇÃO

Dentre os 5 sentidos que possuímos, a visão é o mais avançado e importante de todos (GONZALES, 2002) (RUSS, 2007). É por meio da visão que recebemos a maior parte das informações do ambiente que nos cerca. Esta maior tendência de interação a partir de imagens acaba por influenciar a forma como a informação é representada e armazenada. Instrumentos científicos geralmente produzem imagens para apresentar seus resultados ao operador, ao invés de gerar tons audíveis ou emitir um odor. Da mesma maneira, o sucesso de missões exploratórias de alta demanda científica e econômica, como as espaciais ou ao fundo do mar, geralmente é medido em função da qualidade das imagens obtidas (RUSS, 2007).

Dada a importância do assunto, não por acaso as pesquisas no intuito de executar tarefas relacionadas à visão de modo automático iniciaram-se concomitantemente ao surgimento dos primeiros computadores digitais, na década de 60. Neste contexto, tais tarefas dividem-se nas áreas de (a) melhoramento de imagens para interpretação humana e (b) processamento de imagens para armazenamento, transmissão e percepção artificial (GONZALEZ, 2002). Embora o interesse em ambas as áreas continue elevado, é no campo da percepção artificial de imagens que atualmente se concentram os maiores esforços. A principal razão para tal reside no fato do grande desafio que tarefas como essa representam de um ponto de vista computacional. Enquanto a percepção visual consiste num processo trivial quando realizado por um humano, o mesmo não se aplica em relação à máquina, justamente pela dificuldade de modelar os processos cognitivos empregados pelo humano na realização da tarefa. Neste sentido, métodos baseados em inteligência artificial, como classificadores estatísticos, Redes Neurais Artificiais (RNAs), classificadores sintáticos e sistemas nebulosos (SONKA, 2007) têm sido usados na tentativa de modelagem dos processos biológicos envolvidos na visão.

Outra questão que tange à complexidade dos processos de visão computacional diz respeito à elevada demanda de processamento exigida pelos mesmos. O fato das operações serem efetuadas em nível de *pixel* e muitas vezes abrangerem a imagem como um todo, consistem na principal razão dessa elevada demanda. Quando não é possível a substituição de um dado método por outro de menor demanda computacional, o desempenho de tempo real só é alcançado por intermédio de *hardware* dedicado à realização daquela tarefa (SONKA, 2007).

1.1 Motivação

A importância e as implicações computacionais da visão artificial, aliada à evolução experimentada recentemente pela computação configurável, criam um ambiente favorável para a pesquisa e desenvolvimento de sistemas heterogêneos de visão artificial. Não por coincidência, computadores embarcados e sistemas de visão computacional são dois dos maiores mercados em expansão na indústria. Neste contexto, sistemas em *chip* (SoCs) têm experimentado uma popularidade crescente, em vista da economia no consumo de potência, custos financeiros e área proporcionada pelos mesmos. A partir da integração de componentes (inclusive analógicos) em um único *chip* ocorre a viabilização (e até a preferência) da solução de novos e antigos problemas de cunho industrial, militar e do mercado de consumo por meio de sistemas de visão computacional (KÖLSCH, 2007).

Embora um ambiente que enderece de forma eficiente todas as etapas envolvidas no co-projeto de *hardware* e *software* ainda não seja uma realidade (OU, 2005) (HA, 2006), as metodologias e ferramentas atuais já possibilitam o desenvolvimento de sistemas embarcados de uma forma muito mais integrada. A abordagem de desenvolvimento original, baseada em fluxos divergentes de *hardware* e *software*, tem dado cada vez mais espaço a metodologias que integram tais fluxos, o que aumenta a eficiência do processo.

Aliada ao avanço das metodologias e ferramentas, tem ocorrido também a evolução dos dispositivos e das plataformas de implementação de sistemas embarcados. Neste contexto, os *Field Programmable Gate Arrays* (FPGAs) foram promovidos de meros coadjuvantes a atores principais. Historicamente, tais dispositivos têm sido utilizados na implementação de lógica de ligação entre os demais dispositivos de um sistema. Entretanto, fatores como o aumento da densidade lógica, a incorporação de elementos como memórias, multiplicadores e principalmente microprocessadores (*hard* e/ou *soft cores*) elevaram o *status* dos FPGAs de blocos construtores para plataforma de implementação (DAVIS, 2005) (FLETCHER, 2005).

1.2 Objetivos

A partir das considerações efetuadas nas seções anteriores, nota-se a franca expansão de ambas as áreas de sistemas embarcados e visão computacional, bem como as ótimas possibilidades que surgem pela convergência das mesmas. A vasta gama de aplicações onde visão computacional pode ser utilizada, entretanto, impossibilita o emprego de uma única alternativa na solução do problema como um todo.

Assim sendo, a proposta deste trabalho consiste no desenvolvimento de uma solução embarcada que abrange a área de reconhecimento de caracteres ópticos. Tal solução compreende uma arquitetura heterogênea, desenvolvida a partir do particionamento das tarefas em elementos de *hardware* e *software*, capaz de executar funções referentes à extração e ao reconhecimento de caracteres ópticos presentes em imagens digitais.

Dentre os sistemas que empregam tais funções, estão os Sistemas de Reconhecimento de Placas de Licenciamento Veicular (SRPLVs), utilizados como estudo de caso neste trabalho. SRPLVs são geralmente constituídos pelas etapas de localização de placa, localização de caracteres e reconhecimento de caracteres. Neste sentido, o intuito deste trabalho é o de implementar funções referentes às duas últimas etapas. No que diz respeito às aplicações de um SRPLV, a identificação da placa pode ser utilizada de várias formas, como controle de acesso, controle de tráfego, fiscalização, etc. Entretanto, a arquitetura aqui proposta pode ser empregada em outras aplicações que compartilhem da mesma problemática, como no caso de identificação do conteúdo de placas de trânsito (MOLZ, 2001) e identificação de peças numa linha de montagem, a partir de alguma informação alfanumérica presente na mesma, como um número de série, por exemplo.

Desta maneira, além do desenvolvimento da arquitetura em si, outros objetivos pretendidos por este trabalho consistem na comparação de desempenho entre um modelo de referência puramente descrito em *software* e o sistema heterogêneo, nas implicações do emprego de uma unidade de ponto flutuante integrada ao microprocessador, bem como no uso de *hard* e *soft processors*, além da análise de métodos de co-projeto de *hardware* e *software*.

1.3 Metodologia

Inicialmente, uma versão totalmente em *software* de um SRPLV foi construída sobre uma plataforma x86 (*Windows XP*), com o auxílio da ferramenta *C++ Builder*, versão 6, fornecida pela empresa *Borland Software Corporation*. O intuito da construção de tal versão era tanto o de selecionar os algoritmos mais propícios para a implementação em *hardware*, como de possuir um parâmetro para a posterior comparação de desempenho com a versão embarcada. Conforme mencionado na seção anterior, o escopo deste trabalho se restringe às etapas de localização e reconhecimento de caracteres de um SRPLV, tendo a etapa de localização da placa sido foco de outro trabalho correlacionado (PACHECO, 2007). Assim sendo, os dados de entrada utilizados por este trabalho consistem na saída disponibilizada pelo trabalho desenvolvido por Pacheco (2007). Tais dados consistem nas coordenadas de um

retângulo pertencente a uma imagem contendo um veículo em trânsito, retângulo este no qual há uma grande probabilidade de estar situada a placa deste veículo.

Após o término da implementação da versão de *software* sobre plataforma x86, a validação preliminar da mesma deu-se a partir de uma base de 791 imagens. A partir dessa validação, foram selecionados os métodos propícios para implementação em *hardware*. No que tange o algoritmo de treinamento da RNA utilizada na etapa de reconhecimento de caracteres, o mesmo teve inicialmente uma implementação efetuada em *software*. Entretanto, por conta da vasta disponibilidade de algoritmos de treinamento do pacote de RNAs que compõe o ambiente MATLAB, versão R2006a, fornecido pela empresa *The MathWorks, Inc.*, o mesmo foi utilizado posteriormente nesta tarefa.

O próximo passo consistiu na implementação dos blocos *hardware*, tarefa esta realizada com o auxílio de inúmeras ferramentas fornecidas pelas empresas *Mentor Graphics* e *Xilinx, Inc.*, as quais são detalhadamente especificadas no Capítulo 5. O desenvolvimento dos blocos de *hardware*, bem como do sistema embarcado em si, foi baseado numa placa de prototipação XUP V2-Pro. Esta placa é composta por inúmeros periféricos e um FPGA XC2VP30-FF896, o qual possui dois *hard cores PowerPC* 405 embutidos, dentre outros elementos. A validação física dos módulos de *hardware* foi efetuada com auxílio de uma opção específica para este fim, desenvolvida no sistema sobre plataforma x86, sendo a comunicação entre o *Personal Computer* (PC) e a placa efetuada por intermédio de uma conexão *Recommended Standard*-232 (RS-232).

Após a construção e validação dos módulos de *hardware*, foi efetuada a elaboração do sistema embarcado básico. Neste momento foram especificados elementos como o sistema operacional, o microprocessador, os periféricos e os respectivos *drivers* que formam a base do sistema embarcado. Imediatamente após a construção do sistema base, realizou-se o mapeamento da RNA implementada sobre plataforma x86 para a plataforma embarcada. Como os microprocessadores de ambas plataformas possuíam compiladores para a linguagem C++, a migração foi efetuada sem a existência de maiores problemas. Ainda durante esta etapa foram efetuadas análises sobre o impacto da utilização de *hard* e *soft cores*, bem como de unidade de ponto flutuante junto to microprocessador.

Finalmente os blocos de *hardware* foram consolidados junto ao sistema embarcado base, agora composto também pelo *software* embarcado de RNA. A comunicação entre os blocos de *hardware* e o *software* embarcado foi efetuada por intermédio da escrita de registradores de acesso comum, método este que consiste na maneira mais simples de comunicação entre os elementos heterogêneos. Algumas validações pontuais, como a

referente à transmissão de dados entre o PC e o FPGA, foram realizadas com auxílio de um analisador lógico de 64 canais.

Para a comparação de desempenho entre o sistema sobre plataforma x86 e o sistema embarcado, uma opção específica para este fim foi incorporado ao sistema sobre plataforma x68. Desta vez foi utilizada uma base de 3.000 imagens, sendo que a primeira providência foi a classificação manual das mesmas (com auxílio dos métodos implementados no sistema sobre plataforma x86), no intuito de apurar as respostas esperadas dos sistemas. Em seguida, o processamento foi repetido, mas dessa vez de forma automática, primeiramente pelo sistema sobre plataforma x86 e após, pelo sistema embarcado, via conexão RS-232. Posteriormente, as respostas e os tempos de processamento fornecidos pelos dois sistemas para cada imagem foram sintetizados e comparados.

1.4 Contribuições do Trabalho

A partir do estudo realizado neste trabalho acerca dos conceitos e métodos relativos principalmente às áreas de visão computacional e sistemas embarcados, têm-se as seguintes contribuições como as mais importantes:

- A definição e implementação de uma arquitetura embarcada comum às aplicações que endereçam a problemática do reconhecimento de caracteres ópticos;
- A criação de uma biblioteca de *software* composta por uma série de métodos relativos à área de processamento digital de imagens, como binarização global e local, etiquetação de componentes conectados, projeção horizontal e vertical, caixa limitante adaptativa e interpolação “vizinho mais próximo”;
- A implementação da arquitetura de *hardware* equivalente para alguns dos métodos que compõem a biblioteca de *software*, como o de etiquetação de componentes conectados e o de interpolação “vizinho mais próximo”;
- O mapeamento do algoritmo de propagação de uma RNA *perceptron* de múltiplas camadas (PMC) que emprega aritmética de ponto flutuante para a sua versão equivalente em ponto fixo;
- A definição de um método eficaz para a realização do particionamento dos elementos de *hardware* e *software* num sistema embarcado, a partir de um fluxo de desenvolvimento não integrado.

1.5 Organização do Trabalho

Este trabalho está organizado em sete capítulos. No capítulo seguinte são abordados conceitos referentes à área de visão computacional. Neste contexto, são considerados desde definições sobre imagens digitais até técnicas de inteligência artificial para o reconhecimento de padrões, como é o caso de RNAs. Ainda neste capítulo são apresentados detalhes sobre um SRPLV, o qual consiste no estudo de caso deste trabalho. Encerrando a parte do trabalho que trata das definições teóricas, o Capítulo 3 discute os conceitos e evoluções referentes a sistemas embarcados, bem como a metodologia de desenvolvimento adotada para os mesmos, denominada co-projeto de *hardware* e *software*. Em seguida, o Capítulo 4 apresenta a proposta de uma arquitetura heterogênea voltada à extração e reconhecimento de caracteres ópticos, onde são discutidos os métodos considerados e as escolhas realizadas. Na seqüência, a implementação da arquitetura proposta é discutida em detalhes no Capítulo 5 pela apresentação dos resultados parciais obtidos, das dificuldades encontradas e as soluções adotadas durante o processo de desenvolvimento. No Capítulo 6 são então apresentados os resultados obtidos a partir da comparação efetuada entre os sistemas implementados sobre plataforma x86 e embarcada, ressaltando as vantagens e desvantagens de cada uma delas. Finalmente, as considerações finais, bem como a definição de trabalhos futuros, são realizadas no Capítulo 7.

2 VISÃO COMPUTACIONAL

2.1 Introdução

Conforme mencionado no capítulo introdutório, o processamento digital de imagens, no sentido mais amplo do termo, consiste numa área vasta, que engloba tarefas de diferentes complexidades. Neste sentido, convém inicialmente esclarecer pontos como as suas origens, os conceitos envolvidos e algumas das técnicas empregadas. Dentre tais técnicas, uma seção inteira foi dedicada à revisão de RNAs, empregadas como mecanismo de reconhecimento neste estudo. Após tais considerações, são apresentados os conceitos relativos a SRPLVs, aplicação esta que foi objeto de estudo deste trabalho.

2.2 Definições

Segundo Gonzalez (2002), o campo denominado processamento digital de imagens diz respeito ao processamento de imagens digitais por meio de computadores digitais. O interesse por esse campo é motivado por duas principais áreas de aplicação: (a) o melhoramento de informações pictóricas para a interpretação humana e (b) o processamento de dados visuais para armazenamento, transmissão e representação voltada à percepção artificial autônoma.

No tocante à primeira área de aplicação citada, o emprego de tal tecnologia dá margem à extensão da faixa de visão natural. Enquanto os sistemas biológicos estão restritos à banda visível do espectro eletromagnético (ondas de comprimento entre 400 e 700 nanômetros), os sistemas digitais são capazes de cobrir quase todo o espectro, abrangendo desde raios gama até as ondas de rádio.

Já em relação à segunda área de aplicação supracitada, um dos primeiros sistemas a empregar imagens digitais foi o *Bartlane*, no início da década de 20. Tal sistema permitia o envio de fotos entre Londres e Nova Iorque por meio de um cabo submarino, sendo a indústria jornalística o principal usuário de tal sistema. Apesar da utilização de imagens digitais pelo sistema *Bartlane*, ele não é de fato considerado como um sistema de processamento digital de imagens, já que computadores digitais não eram utilizados pelo mesmo (até porque, nem existiam naquela época).

O surgimento do que é conhecido hoje como processamento digital de imagens ocorreu de fato em meados da década de 60, quando os primeiros computadores digitais com

poder de processamento de armazenamento suficiente para este fim se tornaram disponíveis. Um dos primeiros trabalhos realizados neste contexto foi o melhoramento de imagens obtidas da superfície lunar pela sonda espacial *Ranger 7*, em 1964, as quais apresentavam distorções causadas pelo tipo de câmeras utilizadas a bordo.

O desenvolvimento da área de percepção artificial autônoma, atualmente denominada visão computacional, ocorreu em concomitância com os demais métodos de aprimoramento de imagens digitais. O progresso obtido no início da década de 70 na área de inteligência artificial foi utilizado como subsídio por Marr, que propôs o primeiro modelo de visão computacional no início da década de 80 (ALOIMONOS, 1991). Como principal objetivo, um sistema de visão computacional visa emular eletronicamente as capacidades de percepção e entendimento de imagens proporcionadas por um sistema de visão biológico (SONKA, 2007).

Por conta de não haver um consenso na definição das fronteiras entre processamento digital de imagens, visão computacional e outras áreas correlatas, como análise de imagens, costuma-se adotar um paradigma que classifica os métodos em níveis de processamento. Os métodos classificados no nível inferior operam sobre uma imagem no seu formato mais elementar (elementos pictóricos), enquanto que os de nível superior empregam modelos formais para “perceber” elementos contidos nas representações fornecidas pelo(s) nível(is) inferior(es). Conforme Gonzalez (2002), são três esses níveis: baixo, intermediário e alto. Já Sonka (2007) considera a existência de apenas dois níveis de processamento, baixo e alto, sendo que as tarefas de nível intermediário são tidas como de nível baixo.

O processamento de baixo nível diz respeito à aquisição da imagem em si, bem como às operações primitivas de pré-processamento que são realizadas sobre essa imagem. Como exemplos de métodos de pré-processamento podem ser citados os melhoramentos (ajuste de histograma, filtragem para remoção de ruídos, etc.), as transformações (translação, rotação, escala), e a compressão (com ou sem perdas). Neste nível de processamento, tanto a entrada, quanto à saída, consistem numa imagem (matriz de *pixels*).

Já o processamento de nível intermediário envolve as tarefas de segmentação, bem como representação e descrição. A segmentação consiste em separar a imagem em diferentes regiões, sendo que os métodos comumente empregados são os baseados em contorno (transformada de Hough, perseguição de bordas, etc.), baseados em região (crescimento de regiões, separação de regiões, junção de regiões, etc.) e a binarização (global, local, etc.). Depois de segmentadas, as regiões contidas na imagem são geralmente identificadas por um processo denominado etiquetação de componentes conectados e após descritas numa forma mais apropriada para o seu processamento no próximo nível. Como formas de representação

utilizadas na descrição das regiões podem ser citadas as cadeias de códigos, aproximações polinomiais, assinaturas, segmentos de contornos e os *skeletons*.

Finalmente, o processamento de nível alto compreende as tarefas de reconhecimento e interpretação. O reconhecimento consiste na classificação das regiões apuradas nas etapas anteriores, de acordo com características distintas de cada tipo de região, bem como do grupo de regiões em questão. Neste processo de reconhecimento são empregados métodos estatísticos (classificadores de distância mínima, etc.), sintáticos (autômatos) e conexionistas (RNAs). A interpretação, por sua vez, representa um desafio ainda maior que o reconhecimento, pois visa entender o contexto geral apresentado numa imagem, a partir dos elementos que a compõem. Métodos como lógica formal, redes semânticas e sistemas de produção costumam ser empregados neste caso.

2.2.1 Aplicações

Atualmente o processamento digital de imagens é empregado numa vasta gama de aplicações. São poucas as áreas técnicas que ainda não lançam mão desta técnica. Dentre as inúmeras áreas e respectivas aplicações, se pode citar as seguintes:

- Medicina
    - Detecção de patologias dos ossos e Tomografia por Emissão de Pósitrons (PET) – imagens geradas por raios gama;
    - Melhoramento do contraste de “chapas” de raios-X ou angiogramas – imagens geradas por raios-X;
    - Construção de imagens em 2D a partir do emprego de ressonância magnética – imagens geradas por ondas de rádio.
- Astronomia
    - Detecção de estrelas pela análise da radiação gama gerada quando da explosão de uma – imagens geradas por raios gama;
    - Identificação de estrelas pela análise da radiação X, ultravioleta e das ondas de rádio emitida pelas mesmas – imagens geradas por raios-X, ultravioletas e ondas de rádio, respectivamente;
- Indústria
    - Inspeção automática no intuito de localizar falhas de processo, como a falta de componentes, preenchimento insuficiente de embalagens,

anomalias de cor, etc. – imagens geradas por raios-X, ondas visíveis e infravermelhas;

- Microscopia
  - Identificação de espécimes a partir da técnica de fluorescência – imagens geradas por raios ultravioletas;
  - Micro-inspeção, caracterização de materiais, etc. – imagens geradas por ondas visíveis e infravermelhas;
- Radar
  - Permite a coleta de dados sobre virtualmente qualquer área, independente de horário, condição do tempo ou iluminação – imagens geradas por microondas;
- Outras aplicações utilizando imagens geradas por ondas visíveis ou infravermelhas
  - Reconhecimento automático de impressões digitais, cédulas monetárias, placas de licenciamento veicular, etc.;
  - Sensoriamento remoto para mapeamento temático da superfície terrestre e previsão do tempo;

2.3 Fundamentos relevantes sobre imagem digital

2.3.1 Definição

Uma imagem pode ser definida como uma função bi-dimensional contínua, $f(x, y)$, onde x e y são as coordenadas espaciais (num plano) e a amplitude de f em qualquer par (x, y) é chamada de intensidade ou nível de cinza da imagem naquele ponto (GONZALEZ, 2002). Uma imagem é dita digital quando as coordenadas x e y, bem como a amplitude f, são todos representados por quantidades discretas. Por sua vez, uma imagem digital é composta por um número finito de elementos, cada um com uma localização distinta e valor. Tais elementos são denominados *elementos pictóricos*, *elementos da imagem*, *pels* ou *pixels*, sendo que a última terminologia é a mais comumente adotada.

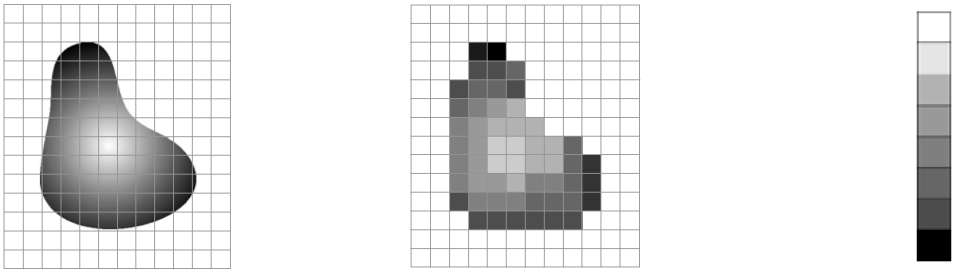
Por convenção e conforme ilustrado na Figura 2.1, a referência espacial (origem) deste sistema de coordenadas se localiza no canto esquerdo superior da imagem, sendo que o valor da coordenada x aumenta da esquerda para a direita, ao passo que o da coordenada y, de cima para baixo (NI, 2007).

Figura 2.1 – Ilustração de uma imagem digital, composta por uma coleção de elementos discretos *f(x,y)* e definida num plano cuja origem se situa no canto superior esquerdo.

2.3.2 Amostragem e quantização

Por conta de uma imagem ser definida por uma função contínua *f(x, y)*, a mesma deve ser mapeada para o formato digital antes que um computador possa realizar qualquer tipo de processamento sobre ela. Tal processo de mapeamento é denominado digitalização, sendo composto pelas etapas de amostragem e quantização.

Considerando a utilização de uma matriz de sensores no processo de digitalização, a Figura 2.2 ilustra a situação onde uma imagem contínua se encontra projetada nesta matriz. Em seguida, a amplitude contínua de cada um dos elementos da matriz é apurada (amostragem) e transformada num valor discreto equivalente (quantização), dando origem à imagem exibida na Figura 2.2 (b). Em outras palavras, quantizar um elemento significa apurar um valor discreto para o mesmo, a partir do mapeamento entre o valor contínuo armazenado nele e uma escala pré-definida de níveis, como a paleta de oito tons de cinza usada neste exemplo (Figura 2.2 (c)).



(a) (b) (c)

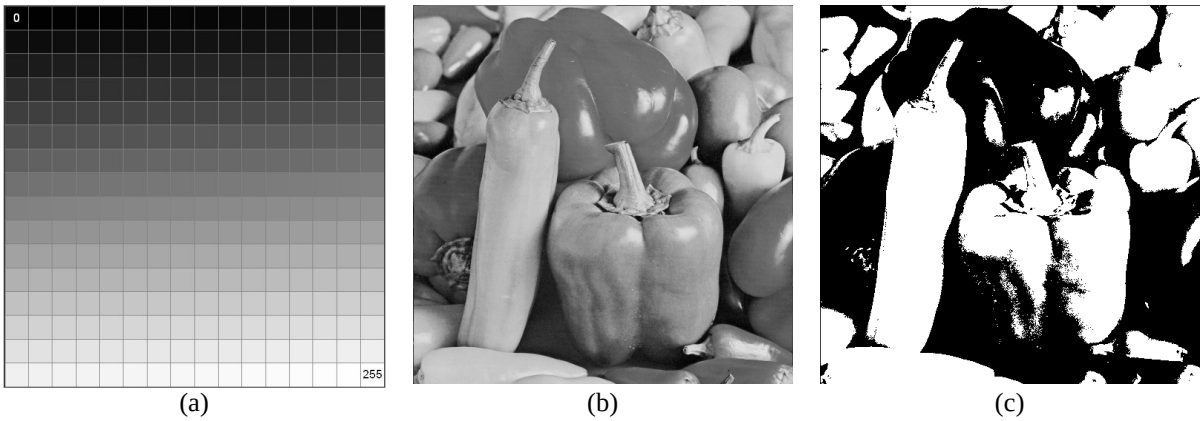
Figura 2.2 – Ilustração de (a) uma imagem contínua projetada numa matriz de sensores e (b) sua versão amostrada e quantizada, considerando (c) uma paleta de cores composta por oito tons de cinza.

Obviamente, a qualidade de uma imagem digital é influenciada pela forma como ela é amostrada. Quanto maior a densidade de amostras digitais, maior o nível de detalhamento que essa imagem pode exibir. Da mesma maneira, a qualidade também é afetada pelo número de níveis de quantização utilizado, que é dado em função do número de *bits* por *pixel* (SONKA, 2007).

2.3.2.1 Tipos de imagens

De um ponto de vista de número de *bits* por *pixel*, as imagens digitais costumam ser classificadas em pelo menos dois tipos: níveis de cinza e colorida.

As imagens em níveis de cinza costumam utilizar um *byte* (ou oito *bits*) para representar cada *pixel*. Assim sendo, empregam uma paleta de 256 níveis de cinza, como a ilustrada na Figura 2.3 (a), onde o tom mais escuro é atribuído ao índice 0 e o tom mais claro, ao índice 255, dando origem à imagens como a exibida na Figura 2.3 (b). A Figura 2.3 (c) ilustra o que pode ser considerado como um caso particular de imagens em níveis de cinza: uma imagem binária. Imagens binárias utilizam apenas um *bit* por *pixel*, sendo que deste modo, somente duas cores (ou níveis de cinza) são adotadas: branca e preta (ou o nível mais claro e o nível mais escuro). Entretanto, a classificação de imagens binárias como um caso específico das imagens em níveis de cinza não é um consenso, podendo as mesmas designar outra classe denominada *bitmap*.



(a) (b) (c)

Figura 2.3 – Ilustração de (a) uma paleta de cores composta por 256 tons de cinza, utilizada para representar a imagem em (b). Já em (c), a mesma imagem foi representada com apenas os dois tons desta paleta (0 e 255).

Já as imagens coloridas costumam empregar três amostras para representar cada *pixel*. Cada amostra diz respeito a um canal de cor distinto, sendo que os modelos de cores mais

utilizados são o *Red, Green, Blue* (RGB) e o *Cian, Magenta, Yellow, Key (black)* (CMYK). Geralmente cada amostra (ou canal) é representada por oito *bits*, totalizando assim 24 *bits* por *pixel*, ou 16.777.216 diferentes tons. Não há nenhum impedimento, entretanto, na utilização de mais de oito *bits* por canal, tampouco de mais canais por *pixel*. A Figura 2.4 ilustra uma imagem RGB de 24 *bits* por *pixel*.

Figura 2.4 – Exemplo de uma imagem colorida no formato RGB de 24 *bits* por *pixel*.

2.3.2.2 Binarização

Dependendo do método empregado no processamento de uma imagem digital, necessita-se que a mesma esteja representada em um determinado formato (tipo). Enquanto a conversão de uma imagem colorida, no formato RGB de 24 *bits*, para a sua versão equivalente em 256 tons de cinza pode ser efetuada pela apuração da média dos três tons que compõem cada *pixel*, o mapeamento para o formato binário não é igualmente trivial.

O processo de binarização geralmente é composto por duas etapas e efetuado a partir de uma imagem em 256 tons de cinza. A primeira etapa consiste na apuração de um valor limiar, o qual é utilizado na segunda etapa para definir o valor de cada *pixel* da imagem resultante. Caso o valor do *pixel* da imagem original seja superior ao do limiar, ele é considerado como sendo da cor branca na imagem resultante, caso contrário, a cor preta é atribuída ao mesmo. Neste sentido, dois métodos populares na realização deste processo são o de Otsu e o de Niblack.

O método de Otsu é dito global por considerar todos os *pixels* da imagem na busca do valor limiar. Segundo Tian (2003), um valor ótimo para tal limiar é obtido a partir de

$$t^* = Arg\left\{ \max_{0 \le i \le L} \left(\frac{\sigma_B^2}{\sigma_T^2} \right) \right\} \qquad (2.1)$$

pela maximização da relação entre a variância inter-classes e a variância total, sendo a primeira definida por

$$\sigma_B^2 = \frac{(\omega_0 \mu_T - \mu_t)^2}{\omega_0 \omega_1} \tag{2.2}$$

e a segunda definida por

$$\sigma_T^2 = \sum_{i=0}^{L-1} (i - \mu_T)^2 \frac{n_i}{N} \tag{2.3}$$

onde $\omega_0 = \sum_{i=0}^{t} \frac{n_i}{N}$, $\omega_1 = 1 - \omega_0$, $\mu_t = \sum_{i=0}^{t} i \frac{n_i}{N}$, $\mu_T = \sum_{i=0}^{L-1} i \frac{n_i}{N}$, n_i é o número de *pixels* que possuem o nível de cinza *i*, *L* é o número de níveis de cinza e *N* é o número total de *pixels* na imagem.

Já o método de Niblack é tido como local, por basear o processamento no emprego de uma janela deslizante. Neste caso o limiar é calculado a partir dos *pixels* contidos nesta janela, a qual deve ser pequena o suficiente para preservar os detalhes locais, mas ao mesmo tempo grande o suficiente para suprimir eventuais ruídos (LEEDHAM, 2003), sendo tal limiar definido por

$$t^* = \bar{x} + k\sigma \tag{2.4}$$

onde $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$ (média), $\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2}$ (desvio padrão), *k* é uma constante que define o percentual de utilização dos *pixels* nos limites da imagem e *N* é o número total de *pixels* contido na janela utilizada.

2.3.2.3 Redimensionamento

O redimensionamento de imagens é um tópico relacionado ao processo de amostragem e quantização, pois a ampliação pode ser vista como sobre-amostragem e a redução como sub-amostragem. A principal diferença consiste no emprego de amostragem e quantização no caso de uma imagem original contínua, enquanto que ampliação e redução são realizadas numa imagem digital (GONZALEZ, 2002).

Quando uma imagem é redimensionada, cada elemento da nova matriz de destino acaba por ocupar um espaço menor que é ocupado por um elemento na matriz de origem, no caso de uma ampliação, ou um espaço maior que o original, no caso de redução. Essa diferença de escala acaba por gerar coordenadas não inteiras durante o processo de mapeamento da imagem original para a nova imagem.

Vários métodos são empregados na solução deste problema, sendo que o mais simples consiste no descarte da parte fracionária do endereço calculado (truncagem), fazendo com que o *pixel* mais próximo do endereço original, na direção da origem do sistema de coordenadas, seja usado. Outro método de complexidade semelhante, mas com resultado ligeiramente superior é o chamado interpolação *nearest neighbor* (“vizinho mais próximo”, em português). Ao invés de truncar, este método arredonda o valor do endereço calculado, procedimento este que reduz o erro de aproximação ao endereço original, se comparado com o método anterior (RUSS, 2007) (PRATT, 2007a).

Outros dois métodos populares de redimensionamento são a interpolação bilinear e a interpolação bicúbica. Enquanto o primeiro apura a média das intensidades dos quatro vizinhos mais próximos ao endereço original, o segundo efetua uma interpolação ponderada, utilizando coeficientes gerados a partir da parte real dos endereços calculados e levando em consideração uma vizinhança de oito *pixels*.

A Figura 2.5 exibe uma comparação de resultados, obtidos a partir do emprego dos três métodos supra-citados na ampliação em 32 vezes de uma imagem de 64 x 64 *pixels*. Na Figura 2.5(a), é notável o serrilhamento causada pelo método “vizinho mais próximo”, enquanto que diferenças sutis podem ser notadas entre os métodos bilinear e bicúbico, principalmente nas áreas de maior diferença de contraste, conforme ilustrados na Figura 2.5(b) e na Figura 2.5(b), respectivamente.

(a) (b) (c)

Figura 2.5 – Resultados obtidos a partir da ampliação em 32 vezes de uma imagem de 64x64 *pixels* pelos métodos de interpolação (a) “vizinho mais próximo”, (b) bilinear e (c) bicúbico.

2.3.3 Alguns relacionamentos básicos entre os *pixels*

Apesar do *pixel* ser a unidade elementar de uma imagem digital, existem várias situações onde não apenas ele, mas também aqueles que o cercam devem ser levados em

consideração. Dentre as formas de relacionamento mais importantes entre os *pixels*, estão o conceito de vizinhança e o de conectividade.

2.3.3.1 Vizinhança

Segundo Gonzalez (2002), um *pixel p*, situado na coordenada *(x, y)*, possui quatro vizinhos nas direções horizontal e vertical, cujas coordenadas são dadas por:

$$(x, y-1)$$
$$(x-1,y) \quad p \quad (x+1,y)$$
$$(x, y+1)$$

Este conjunto de *pixels* é chamado de "quatro vizinhos de *p*" e denotado por $N_4(p)$. Cada vizinho está situado a uma distância unitária de *p* e alguns deles residem fora dos limites da imagem digital, quando *p* está posicionado numa das bordas desta imagem.

Além dos vizinhos ortogonais, também existem os quatro vizinhos diagonais de *p*, cujas coordenadas são dadas por

$$(x-1, y-1) \qquad (x+1, y-1)$$
$$p$$
$$(x-1, y+1) \qquad (x+1, y+1)$$

e denotados por $N_D(p)$. A interseção entre $N_4(p)$ e $N_D(p)$ dá origem ao conjunto chamado de "oito vizinhos de p", denotado por $N_8(p)$. Da mesma forma que anteriormente, alguns elementos de $N_D(p)$ e $N_8(p)$ residem fora dos limites da imagem digital quando *p* está posicionado nas bordas da mesma.

2.3.3.2 Conectividade

A conectividade entre *pixels* é um conceito fundamental que simplifica demais conceitos, como os de região e fronteira. Para que exista conectividade entre dois ou mais *pixels*, além de serem vizinhos, também é necessário que seus níveis de brilho satisfaçam um critério pré-definido. Tomando como exemplo uma imagem binária, como a ilustrada na Figura 2.6, apesar dos *pixels q, r, s, t, u, v, x* e *y* estarem situados na vizinhança de *p*, apenas *r, t, v,* e *y* possuem conectividade com *p*, pois satisfazem o critério de possuir o mesmo nível de brilho de *p*.

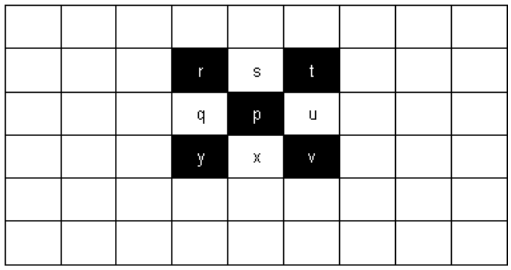


Figura 2.6 – Conectividade entre *pixels* numa imagem binária

Na seqüência são apresentados os conceitos relativos ao método de reconhecimento utilizado por este trabalho: Redes Neurais Artificiais.

2.4 Redes Neurais Artificiais

2.4.1 Introdução

Segundo Arbib (1987 apud HAYKIN, 1999), o sistema nervoso humano pode ser visto como um sistema de três estágios, conforme ilustrado pelo diagrama de blocos na Figura 2.7. No centro deste sistema situa-se o cérebro humano (Rede neural), o qual comunica-se de uma forma interativa com os demais subsistemas. Neste contexto, o subsistema de receptores converte os estímulos externos em sinais elétricos e os repassa adiante, enquanto o subsistema de atuadores transforma os sinais elétricos gerados pelo cérebro em respostas discerníveis de saída do sistema.

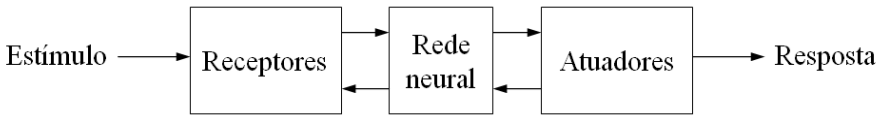


Figura 2.7 – Representação em diagrama de blocos do sistema nervoso humano.

No processo de compreensão da estrutura e funcionamento do cérebro humano, foi de grande relevância a pesquisa pioneira de Ramón e Cajál, publicada em 1910. Dentre os conceitos estabelecidos pelos mesmos está a idéia do neurônio como elemento estruturante do cérebro. Conforme a Figura 2.8, um neurônio é composto por quarto principais elementos: corpo celular, dendritos, axônio e terminais sinápticos. Os neurônios se comunicam por meio de ligações denominadas sinapses, onde os terminais sinápticos de um dado neurônio se “conectam” (não há conexão física na junção sináptica) aos dendritos (zonas receptoras) de

um ou mais neurônios vizinhos. A comunicação entre os neurônios se dá por meio de um processo tanto elétrico, quanto químico. De uma maneira simplificada, o processo de comunicação se inicia pela geração de um sinal elétrico no corpo celular (ou nas proximidades dele). Este sinal elétrico percorre o axônio e, ao chegar nos terminais sinápticos, causa a liberação de uma substância química denominada neurotransmissor. Os neurotransmissores repassam então a informação aos dendritos dos neurônios "conectados" e retornam, geralmente, aos terminais sinápticos de origem.

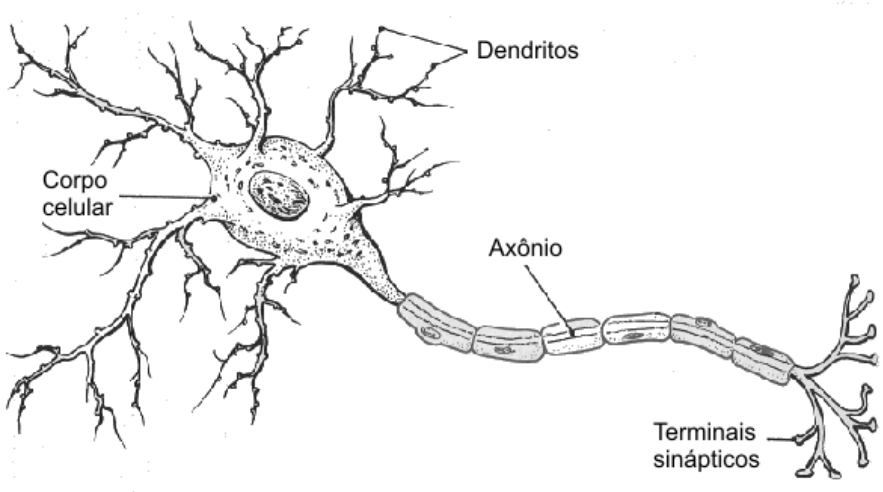


Figura 2.8 – Ilustração de um neurônio biológico e seus quatro principais elementos: corpo celular, dendritos, axônio e terminais sinápticos.

Segundo Shepherd e Kock (1990 apud HAYKIN, 1999), estima-se a existência de mais de 10 bilhões de neurônios no córtex humano, estando os mesmos interconectados por mais 60 trilhões de sinapses. Tal organização faz do cérebro um computador altamente complexo, não-linear e paralelo, possibilitando que o mesmo execute alguns tipos de tarefas (como reconhecimento de padrões, percepção, etc.) de uma forma muito mais eficiente do que um computador eletrônico. Como exemplo desta eficiência pode-se citar a tarefa de reconhecimento de uma face conhecida em um ambiente não-familiar, a qual é executada pelo cérebro num tempo não superior a 200 milésimos de segundo (ms). Tarefas de complexidade inferior podem tomar dias quando executadas num computador convencional (HAYKIN, 1999).

A ciência de que computadores eletrônicos possuem estrutura e funcionamento totalmente distintos de um cérebro humano foi justamente o que motivou o desenvolvimento de pesquisas na área de redes neurais artificiais. De uma forma geral, uma rede neural artificial (comumente denominada por rede neural) é uma máquina projetada para modelar a

maneira como o cérebro humano realiza uma determinada tarefa ou função de interesse. Este tipo de abordagem possibilita a construção de sistemas cuja modelagem é complexa do ponto de vista de métodos convencionais de engenharia. Dentre os elementos que viabilizam este tipo de modelagem, os quais são características inerentes do cérebro humano, pode-se citar os seguintes:

- Não-linearidade: propriedade inerente a sistemas de comportamento indeterminado, ou seja, cuja resposta não pode ser expressa na forma de uma soma linear de componentes independentes que formam o mesmo;
- Mapeamento de entrada/saída: a solução para um problema é determinada por meio do treinamento da rede neural, o qual consiste na apresentação de um conjunto de entradas e as respectivas saídas desejadas. O treinamento é interrompido quando o erro existente entre a saída desejada e a apurada pela rede é menor do que um limite estabelecido;
- Generalização: é a capacidade de um sistema de determinar uma resposta aceitável para uma entrada que ainda não havia sido apresentada ao mesmo, mas que possui semelhança com outras já apresentadas;
- Resposta evidencial: no contexto de reconhecimento de padrões, significa não apenas prover uma resposta para uma determinada entrada, mas também indicar o percentual de certeza sobre a mesma;
- Tolerância à falhas: o fato da informação contida numa rede neural estar dispersa nas inúmeras interconexões existentes entre os neurônios torna praticamente irrelevante o impacto da eventual “perda” de um deles em relação ao funcionamento do sistema.

A pesquisa conduzida por Ramón e Cajál serviu como base na elaboração da teoria sobre redes neurais artificiais, publicada em 1943 por McCulloch e Pitts. Neste trabalho foi demonstrado que, uma rede neural formada por um número suficiente de elementos, com os pesos das conexões ajustados adequadamente e operando de forma síncrona, em princípio, era capaz de implementar qualquer função computável.

Durante a década de 50, a área experimentou um considerável desenvolvimento, a partir de contribuições de nomes como Minsky, Rochester, Holland, Haibt, Duda e Rosenblat. Este último foi responsável pelo desenvolvimento de um novo método de aprendizado supervisionado aplicável ao problema do reconhecimento de padrões, denominado *perceptron*, em 1958.

Já na década de 60, o fato de maior impacto na área foi a publicação do livro *Perceptrons* (1969), por Minsky e Papert. Esta publicação desmentiu matematicamente a falsa impressão que havia surgido no sentido de que as redes neurais eram a solução para todos os problemas. Além disso, também sugeriu que as mesmas limitações enfrentadas pelas redes *perceptron* de camada única se aplicavam às de múltiplas camadas, o que causou uma drástica redução no volume de pesquisas nesta área. Tal suposição foi desmentida apenas em 1986, pela publicação do algoritmo de retropropagação de erro (*backpropagation*), cujos maiores créditos pelo desenvolvimento foram atribuídos a Rumelhart, Hinton e Willians. Nesta mesma década (80), Broomhead e Lowe (1988) propuseram um método alternativo ao *perceptron* multicamadas, denominado *Radial Basis Function* (RBF).

Mais recentemente, na década de 90, Vapnik e colegas desenvolveram uma poderosa classe de redes de aprendizagem supervisionada, denominada *support vector machines*, aplicada na solução de problemas de reconhecimento de padrões (1992), regressão (1995) e estimativa de densidade (1998).

Além dos modelos de redes neurais artificiais aqui mencionados, inúmeros outros existem. Apenas a título de esclarecimento, haja vista que o foco deste trabalho é outro, as redes neurais artificiais existentes podem ser categorizadas da seguinte maneira (NN FAQ, 2008):

- Aprendizado supervisionado
  - *Feedforward*
    - Hebbian, *Perceptron*, Adaline, *backpropagation*, *Orthogonal Least Squares*, etc.;
  - *Feedback*
    - *Bidimensional Associative Memory* (BAM), Máquina de Boltzman, *Finite Impulse Response*, *Time Delay Neural Network*, etc.;
  - Competitiva
    - ARTMAP, *Counterpropagation*, *Neocognitron*, etc.;
- Aprendizado não-supervisionado
  - Competitiva
    - Grossberg, Kohonen, ART1, ART2, *Fuzzy* ART, etc.;
  - Redução de dimensão
    - Hebbian, Oja, etc.;
  - Auto-associativa

- Hopfield, *Brain State in a Box* (BSB), Auto-associador Linear;

- Sem aprendizado
  - Hopfield, várias redes neurais para otimização.

2.4.2 Modelo de neurônio

Um neurônio consiste na unidade de processamento fundamental de uma rede neural. A Figura 2.9 ilustra o modelo neuronal utilizado como base na construção de RNAs, sendo o mesmo constituído por três elementos principais:

- Um conjunto de sinapses (ou conexões), cada uma delas caracterizada por uma entrada x_j que é multiplicada pelo respectivo peso w_k, atribuído à mesma durante o treinamento da rede;
- Um somador que efetua a adição das entradas ponderadas oriundas de cada sinapse, o qual se constitui num combinador linear;
- Uma função de ativação, que limita a faixa de excursão da saída do neurônio, dada pelo somador. Geralmente a saída normalizada fornecida pela função de ativação situa-se no intervalo fechado [0,1] ou [-1,1].

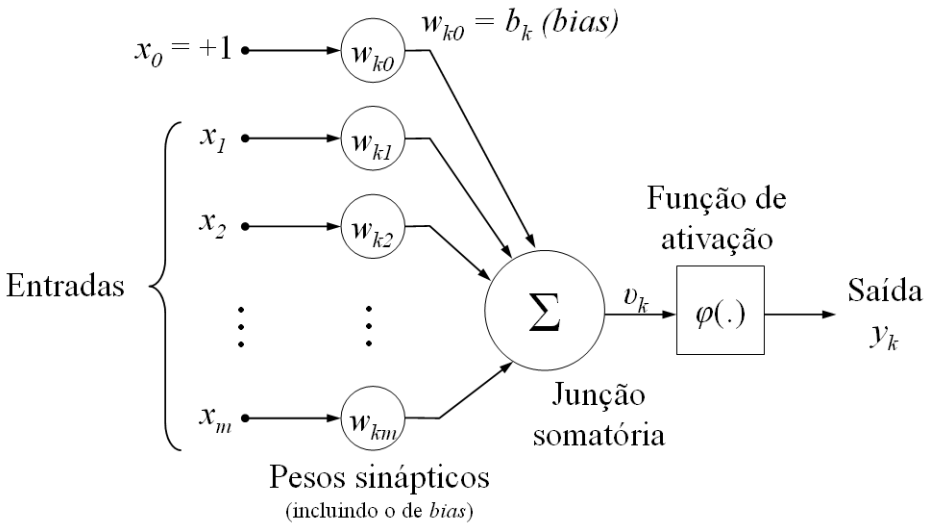


Figura 2.9 – Modelo não-linear de um neurônio.

Além dos três elementos principais, o modelo neuronal inclui também uma conexão externa de *bias*. Tal conexão é composta por uma entrada x_0 de valor fixo +1 (um positivo), bem como um peso w_{k0} dado por b_k. O emprego da conexão de *bias* causa o deslocamento da saída do combinador linear para a banda positiva ou negativa, conforme o valor de b_k.

Assim sendo, o modelo neuronal pode ser expresso matematicamente pelas seguintes equações

$$\upsilon_k = \sum_{j=0}^{m} w_{kj} x_j \quad \textbf{(2.5)}$$

$$y_k = \varphi(\upsilon_k) \quad \textbf{(2.6)}$$

onde x_0 é a entrada fixa +1 da conexão de *bias*; $x_1,x_2,...,x_m$ são os sinais de entrada do sistema; w_{k0} é o peso b_k da conexão de *bias* do neurônio k; w_{k1}, w_{k2}, ..., w_{km} são os pesos das demais sinapses com o neurônio k; υ_k é a saída do combinador linear do neurônio k acrescida do termo referente à sinapse de *bias* (geralmente denominada *campo local induzido*); $\varphi(\cdot)$ é a função de ativação; e y_k é o sinal de saída do neurônio k.

No que diz respeito à função de ativação, a qual é denotada por $\varphi(\upsilon)$ e define a saída do neurônio em função do campo local induzido υ, os tipos mais comumente utilizados são os seguintes:

- Função Limiar: fornece uma saída correspondente à de um neurônio do modelo McCulloch-Pitts, onde a saída assume o valor 1 (um) quando o campo local induzido é não-negativo, ou 0 (zero), caso contrário. Tal função encontra-se ilustrada graficamente na Figura 2.10(a), sendo matematicamente descrita pela seguinte equação

$$\varphi(\upsilon) = \begin{cases} 1, \upsilon \geq 0 \\ 0, \upsilon < 0 \end{cases} \quad (2.7)$$

- Função *Piecewise-linear*: este tipo de função pode ser visto como uma aproximação de um amplificador não-linear. A função se comporta como um combinador linear enquanto a região de operação não alcança a saturação. Quando em saturação, a função responde como uma função limiar. Tal função encontra-se ilustrada graficamente na Figura 2.10(b), sendo matematicamente descrita pela seguinte equação

$$\varphi(\upsilon) = \begin{cases} 1, \upsilon \geq +1/2 \\ \upsilon, +1/2 > \upsilon > -1/2 \\ 0, \upsilon < -1/2 \end{cases} \quad (2.8)$$

- Função Sigmoidal: por conta do balanço exibido entre seus comportamentos linear e não-linear, consiste na mais popular função de ativação utilizada na construção de redes neurais. Um exemplo de função sigmoidal é a função logística, ilustrada na Figura 2.10(c) e definida por

$$\varphi(v) = \frac{1}{1+\exp(-a\,v)} \tag{2.9}$$

onde a é o parâmetro de inclinação. Desta forma, quanto maior o valor de a, mais similar à uma função limiar a função sigmoidal se torna. Entretanto, a segunda assume valores contínuos no intervalo [0,1], sendo portanto classificada como não-simétrica, enquanto a primeira é capaz de assumir apenas valores discretos.

Outro exemplo de função sigmoidal é a tangente hiperbólica, definida por

$$\varphi(v) = \tanh(v) \tag{2.10}$$

a qual assume valores na faixa [-1,1], sendo classificada, portanto, como função anti-simétrica.

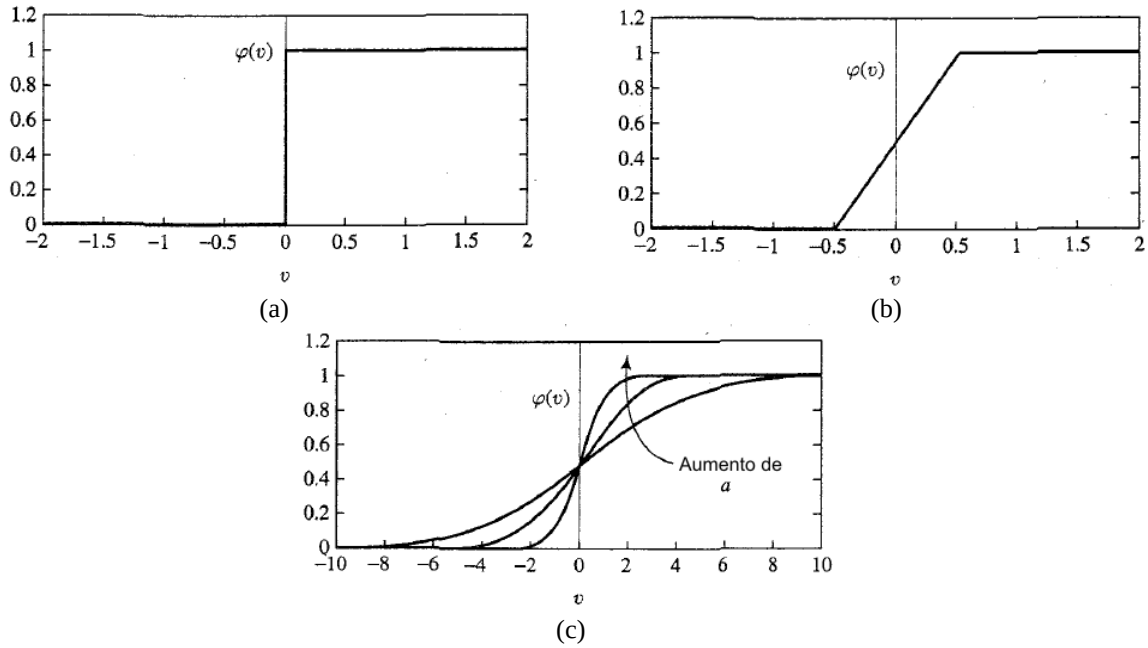


Figura 2.10 – Tipos de função de ativação utilizadas numa RNA (HAYKIN, 1999).

2.4.3 Arquiteturas de rede

A maneira como uma rede neural é estruturada está intimamente ligada com o algoritmo de treinamento usado pela mesma. Em geral, são três os tipos distintos de arquiteturas existentes:

- Redes *feedforward* de camada única: consistem na forma mais elementar de redes *feedforward*, onde a camada de entrada conecta-se diretamente à camada

de saída. Conforme o próprio nome indica, redes *feedforward* são acíclicas, ou seja, as conexões se dão apenas num sentido (da entrada para a saída). Como a camada de entrada não é levada em consideração na contagem do número de camadas, pois nenhum processamento é efetuado na mesma, a Figura 2.11(a) ilustra o caso de uma rede *feedforward* de camada única com quatro neurônios em ambas as camadas de entrada e saída.

- Redes *feedforward* de múltiplas camadas: as redes *feedforward* de múltiplas camadas caracterizam-se pela existência de uma ou mais camadas intermediárias às de entrada e saída. Tais camadas intermediárias são denominadas *camadas escondidas*, sendo seus elementos constituintes denominados *neurônios escondidos*. A inserção de camadas escondidas possibilita que a rede seja capaz de extrair estatísticas de ordem mais elevada, ou seja, permite que a rede adquira uma perspectiva global apesar de sua conectividade local (Churchland e Sejnowski, 1992 apud Haykin, 1999). Numa rede *feedforward* de múltiplas camadas, as entradas de uma camada são sempre dadas pelas saídas da camada anterior, ou seja, o conjunto de saídas dadas pelos neurônios da última camada constitui a resposta geral da rede ao padrão de ativação fornecido pelos nodos fonte na camada de entrada. Um exemplo de uma rede *feedforward* de múltiplas camadas é dado pela Figura 2.11(b), a qual se constitui numa rede de topologia 10x4x2, ou seja, dez nodos de entrada, quatro neurônios escondidos e dois neurônios de saída. No tocante à conectividade desta rede, ela é do tipo *totalmente conectada*, pois cada nodo de uma camada se conecta a todos os nodos da camada seguinte. A rede seria dita *parcialmente conectada* no caso da falta de pelo menos uma sinapse.
- Redes recorrentes: as redes recorrentes diferenciam-se das *feedforward* por conta da existência de pelo menos um laço de realimentação. Tal laço de realimentação conduz as saídas da rede novamente à sua camada de entrada. O laço é dito de auto-realimentação quando a saída de um neurônio alimenta a sua própria entrada. Quanto à existência de camadas escondidas, as redes recorrentes são semelhantes as *feedforward*, podendo ou não tê-las. Assim sendo, a rede ilustrada na Figura 2.11(c) consiste numa rede recorrente de camada única, que não exibe auto-realimentação. A existência de conexões de realimentação melhora a capacidade de aprendizagem e o desempenho da rede,

ao passo que introduz um comportamento dinâmico não-linear, por conta dos operadores de atraso unitário (blocos z^{-1}) utilizados em tais conexões.

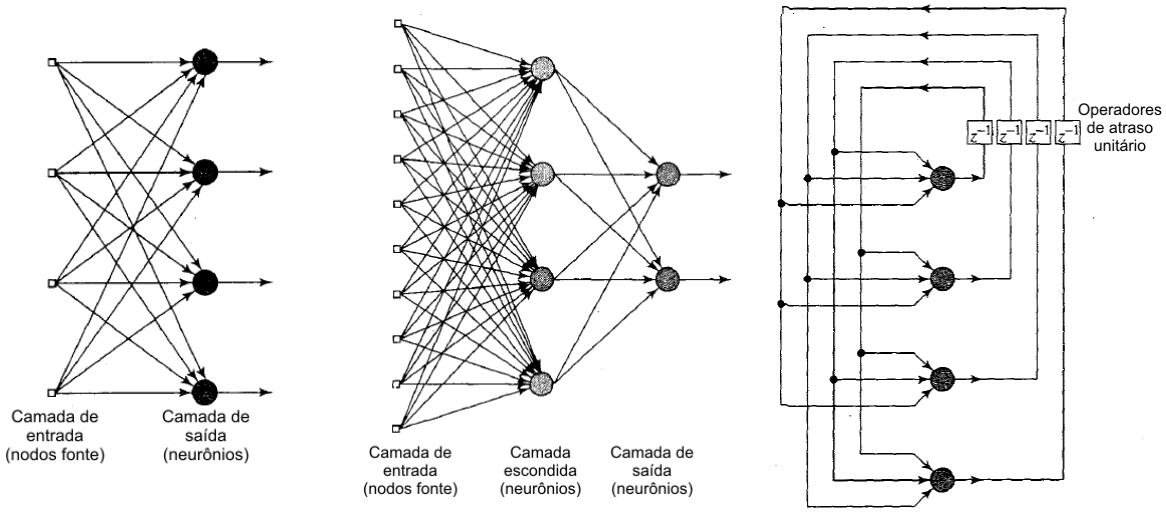


Figura 2.11 – Ilustração das três principais arquiteturas de RNAs: (a) *feedforward* de camada única, (b) *feedforward* de múltiplas camadas e (c) recorrente (HAYKIN, 1999).

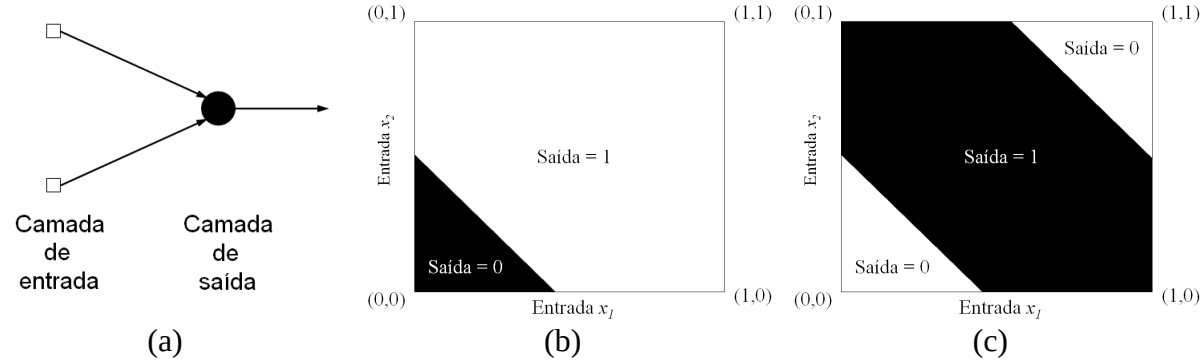
2.4.4 *Perceptrons*

O *perceptron* consiste na forma mais elementar de uma rede neural para classificação de padrões ditos linearmente separáveis. Entretanto, é justamente no fato do algoritmo convergir apenas no caso de classes linearmente separáveis que reside a maior limitação do mesmo, pois é comum a existência de padrões não linearmente separáveis.

Considerando a hipótese de uma rede *perceptron* (de camada única) composta por apenas um neurônio, como a ilustrada na Figura 2.12(a), a mesma é capaz de separar os padrões em duas classes distintas. Um exemplo de problema tratável por uma rede neural como esta é a função lógica *OR* de duas entradas, a qual resulta num conjunto de saídas separáveis por meio de uma linha de fronteira única, conforme a ilustração gráfica exibida na Figura 2.12(b).

Conforme o número de neurônios aumenta, da mesma forma aumenta o número de classes que a rede é capaz de identificar. Entretanto, caso as classes não forem linearmente separáveis na sua totalidade, a rede não será capaz de identificá-las de maneira distinta. O exemplo clássico desta situação consiste na função lógica *XOR*, conforme a ilustração gráfica exibida na Figura 2.12(c). Neste caso, como as entradas que resultam em 0 (zero) (ou seja, (0,0) e (1,1)) residem em cantos opostos do quadrado unitário, são necessárias duas linhas de

fronteira para a correta separação dos padrões de saída. Desta maneira, não há como resolver o problema por meio de uma rede *perceptron* elementar.



(a) (b) (c)

Figura 2.12 – Ilustração de (a) uma rede *perceptron* elementar e das fronteiras de decisão das funções (b) *OR* e (c) *XOR*, sendo a primeira implementável por tal arquitetura, mas não a segunda.

A solução para este problema consiste em tornar a rede neural robusta no que tange a questão da separabilidade dos padrões, ou seja, a rede deve ser capaz de classificar de forma distinta classes não linearmente separáveis. Neste sentido, uma das formas de solução diz respeito à inserção de camadas escondidas na rede, dando origem, assim, a uma rede do tipo *feedforward* de múltiplas camadas. Tais redes são comumente denominadas *perceptron* de múltiplas camadas (PMC).

As redes PMCs tem sido empregadas com sucesso na solução de problemas de resolução complexa. O treinamento de tais redes é realizado de forma supervisionada por meio do algoritmo de retropropagação de erro, o qual é explicado de forma detalhada no Capítulo 4.

Finalizando este capítulo, na próxima seção são apresentados os conceitos relativos a uma das aplicações baseadas em visão computacional: Sistemas de Reconhecimento de Placas de Licenciamento Veicular.

2.5 Sistemas de Reconhecimento de Placas de Licenciamento Veicular

2.5.1 Introdução

Sistemas de Reconhecimento de Placas de Licenciamento Veicular são aqueles que automaticamente identificam um veículo a partir de uma imagem contendo a placa do mesmo (DIAS, 2007). Outras denominações normalmente usadas são as seguintes (HI-TECH, 2007):

- *Automatic Vehicle Identification* (AVI);

- *Car Plate Recognition* (CPR);
- *Automatic Number Plate Recognition* (ANPR);
- *Car Plate Reader* (CPR);
- *Optical Character Recognition* (OCR) *for Cars*.

Os SRPLVs atuais costumam adotar uma estrutura básica comum, diferindo uns dos outros no que diz respeito às técnicas utilizadas na execução das tarefas componentes da mesma. Entretanto, essa maturidade estrutural só foi adquirida cerca de duas décadas após o início das pesquisas, conforme ilustrado na próxima seção.

2.5.2 Histórico

De acordo com Turner (1995), o conceito de SRPLV já era empregado desde o início dos anos 50 para fins de medição de tempo de duração de viagens. Naquela época os registros eram efetuados de forma manual, em um gravador de fita ou mesmo papel, sendo a apuração dos tempos feita posteriormente, com base nestes registros.

Entretanto, como ocorre em toda tarefa manual e repetitiva, não tardou muito para surgir uma solução automática. Os primeiros SRPLVs automáticos datam dos anos 60 (AYLAND,1989), sob a denominação de sistemas AVI. Tais sistemas se baseavam no uso de leitores ópticos e etiquetas de identificação fixadas na área externa dos veículos. Desta forma, o desempenho do sistema era sensível a elementos como neve, chuva, gelo, neblina, sujeita ou mesmo algum problema de alinhamento entre o leitor e a etiqueta.

Leitores infravermelhos foram utilizados na década de 70 numa tentativa de melhorar o desempenho dos sistemas baseados em leitores ópticos. Apesar de apresentarem alguma melhora de desempenho em relação aos seus predecessores, igualmente eram sensíveis às condições climáticas.

Ainda segundo Ayland (1989), o próximo passo na evolução de tais sistemas foi a utilização de tecnologias como laços indutivos, rádio-freqüência e microondas, visando resolver o problema de susceptibilidade ao clima. Dois eram os aspectos em comum destas tecnologias: a utilização de *transponders* afixados no veículo e a forma de classificá-los.

Um *transponder* é um dispositivo eletrônico que emite um sinal de resposta quando solicitado. Este sinal de resposta, neste caso, continha o número de identificação do veículo. Quanto à forma de classificação, podiam ser divididos em ativos, semi-ativos ou passivos, de acordo com a forma como o *transponder* era alimentado.

Nos sistemas do tipo ativo, o *transponder* recebia energia do próprio veículo sob o qual encontrava-se afixado. O sinal de identificação podia ser transmitido continuamente (já que a disponibilidade de energia não era uma questão relevante neste caso), ou então apenas quando solicitado por um receptor.

Já em sistemas passivos, a energia necessária para a transmissão do código de identificação era fornecida pelo próprio laço indutivo, já que os *transponders* não dispunham de fonte de alimentação. Quando fora do campo de alcance do laço indutivo, tais dispositivos encontravam-se totalmente inativos.

Finalmente, sistemas semi-ativos eram aqueles onde os *transponders* dispunham de uma bateria interna própria, transmitindo o código de identificação apenas quando solicitado. A independência de fonte de alimentação externa resolvia os problemas apresentados por um sistema totalmente ativo.

Apesar das primeiras pesquisas sobre o emprego de técnicas de processamento de imagens em sistemas de monitoramento e controle de tráfego datarem do final dos anos 70 (INIGO, 1985), sistemas baseados em laços indutivos continuaram sendo largamente utilizados durante os anos 80 (INIGO, 1989). Entretanto, apesar da desvantagem do ponto de vista econômico à época, o interesse pelo uso de técnicas de processamento de imagens se deu pelo fato destas serem potencialmente mais poderosas e flexíveis que as correntes. Tal colocação pode ser exemplificada por fatos como a necessidade de intervenções no pavimento de uma via para a instalação do sistema ou a impossibilidade de realização de tarefas como detecção de acidentes e rastreamento de veículos, no caso da tecnologia de laços indutivos.

Sistemas de reconhecimento automático de veículos similares aos atuais começaram a surgir no início da década de 90 (LOTUFO, 1990). Os avanços que ocorreram no campo de visão computacional, bem como a queda nos preços dos dispositivos de aquisição de imagens, tiveram forte influência neste contexto. A utilização de técnicas de OCR em substituição aos *transponders* foi favorecida pela disseminação deste tipo de sistemas, haja vista que cada veículo passível de monitoramento necessitava, obrigatoriamente, de um destes dispositivos.

Do início da década passada para cá a estrutura básica dos SRPLVs se manteve praticamente a mesma. As diferenças entre as diversas propostas se situam basicamente nas técnicas empregadas na realização de cada etapa. Deste modo, por exemplo, Nijhuis (1995) melhorou a proposta apresentada por Lotufo (1990) pela redução do tamanho mínimo exigido para a placa do veículo na imagem. Da mesma forma, Barroso (1997) melhorou a proposta de Nijhuis (1995) tornando a etapa de localização da placa independente da cor do fundo da

mesma. Os principais elementos, funcionamento e aplicações de um SRPLV constituem o tema das próximas seções.

2.5.3 Principais elementos

Um SRPLV típico tem como principais elementos os seguintes (DIAS, 2007) (HI-TECH, 2007) (QUERCUS, 2007):

- Sensor de presença: geralmente do tipo indutivo ou capacitivo. É utilizado para ativar o sistema quando da presença de um veículo na área de atuação do mesmo. Não é um item essencial, haja vista que a detecção de presença do veículo pode ser realizada pelo próprio SRLPV;
- Iluminação: utilizada para permitir que o sistema se mantenha operacional na falta ou redução de luz natural. Geralmente é empregada iluminação do tipo infra-vermelha (Figura 2.13), que é imperceptível ao olho humano e permite o realce da zona que contém a placa, a partir de um filtro instalado na câmera;

Figura 2.13 – Exemplos de imagens adquiridas pelo emprego de iluminação infra-vermelha.

- Câmera de vídeo: analógica ou digital. Utilizada para capturar a imagem do veículo e disponibilizá-la ao sistema. No caso de uma câmera analógica, faz-se necessário o uso de uma placa digitalizadora denominada *frame grabber*;
- Computador: um PC convencional, no qual geralmente está conectada a câmera de vídeo e onde reside a aplicação de Reconhecimento de Placas de Licenciamento Veicular (RPLV). Tal aplicação é responsável pelo controle do sistema, leitura das imagens, identificação da placa, bem como pela comunicação com outras aplicações e sistemas;
- *Software*: consiste na aplicação e no pacote de RPLV. Geralmente o pacote é disponibilizado por terceiros como uma biblioteca de *software*, que é agregada a uma aplicação personalizada.

2.5.4 Funcionamento

Conforme mencionado anteriormente, os atuais sistemas de reconhecimento automático de veículos são baseados em visão computacional. Desta forma, não se faz mais necessário o uso de *transponders*, já que o veículo pode ser identificado pela própria placa, por meio do processamento de uma imagem contendo a mesma.

O processo de reconhecimento da placa de licença inicia quando um veículo se aproxima de um ponto de verificação, como, por exemplo, uma cancela, na entrada de um estacionamento público. Assim que a presença do veículo é detectada, seja por um sensor de presença, seja por técnicas de processamento de imagem embutidas no próprio sistema, um sinal de ativação é gerado. Neste momento a câmera adquire uma imagem da parte dianteira ou traseira do veículo e a disponibiliza ao SRPLV, que, por sua vez, localiza a região contendo a placa, separa os caracteres contidos na mesma e os reconhece. Como saída, o *software* de RPLV disponibiliza os caracteres que compõem a placa no formato ASCII. Assim sendo, possibilita a utilização deste dado por qualquer outro processo computacional, desde a simples criação de um registro de entrada, até uma busca em um banco de dados de veículos com infrações pendentes ou roubados.

A Figura 2.14 ilustra o diagrama de blocos básico de um SRPLV. Cada um dos blocos é composto por uma série de técnicas de processamento de imagem, as quais são combinadas de uma forma específica para a resolução da tarefa executada pelo bloco. A mesma técnica pode ser utilizada em mais de um bloco, mas em arranjos distintos para cada bloco. Segue uma descrição detalhada de cada um dos blocos principais.

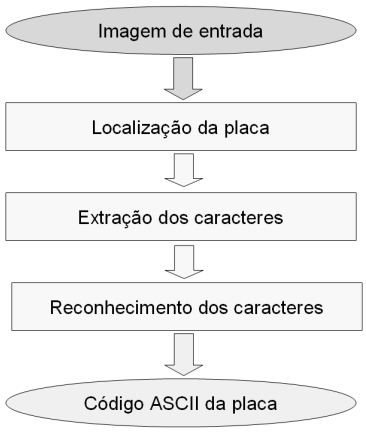


Figura 2.14 – Estrutura básica de um SRPLV.

2.5.5 Localização da placa

Nesta etapa é realizada uma busca por regiões da imagem onde haja uma alta probabilidade de existência de uma placa de licenciamento veicular, como a delimitada pelo retângulo na Figura 2.15. Para tanto, são levadas em consideração características inerentes à mesma, como contraste, textura e geometria.

Figura 2.15 – Região candidata encontrada durante a etapa de Localização da Placa.

Segundo Dias (2005), dentre os métodos empregados para esta tarefa podem ser citados o Método da Morfologia dos Caracteres, o qual se baseia na geometria dos mesmos; o Método da Variação Tonal, que vasculha as linhas em busca de um padrão de freqüência e amplitude (assinatura); o Método da Correlação, que busca correlacionar a sub-imagem do objeto procurado na imagem pesquisada; o Método da Detecção das Arestas, que emprega detectores de contorno para localizar as quatro arestas da placa; o Método Conexionista, emprega técnicas conexionistas na busca por regiões cuja cor e textura sejam similares ao de uma placa de licenciamento veicular; e o Método Híbrido, que emprega uma combinação de dois ou mais métodos distintos, visando aumentar a eficiência do processo.

Esta etapa consiste num ponto crítico do processo de RPLV, pois a não localização de pelo menos uma região contendo a placa de licenciamento veicular implica na falha do processo como um todo.

2.5.6 Extração dos caracteres

Uma por vez, cada região candidata encontrada na etapa anterior é então vasculhada em busca de caracteres componentes de uma possível placa contida na mesma. Dependendo do método empregado no processo de separação dos caracteres, a imagem deve ser previamente binarizada (representada em apenas dois tons de cinza), conforme ilustrado na Figura 2.16. Os caracteres são então separados em função da sua extensão horizontal e vertical, sendo que características geométricas podem ser consideradas para decidir se o objeto em questão é um caractere ou não. Na Figura 2.17, os retângulos vermelhos representam as separações resultantes de um processo de separação.

Figura 2.16 – Ilustração da imagem obtida pelo processo de binarização da região da placa de licenciamento veicular.

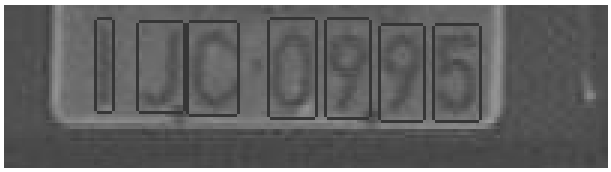


Figura 2.17 – Ilustração das regiões encontradas a partir do processo de localização de caracteres.

Dentre os principais métodos usados nesta etapa, podem ser citados o Método das Projeções (BARROSO, 1997) (MARTÍN, 2002) (DIAS, 2005) , que define os limites de cada caractere por meio de projeções horizontais e verticais; o Método da Morfologia (COWELL, 1995) (COETZEE, 1998) (DIAS, 2005), o qual separa os caracteres com base em seus componentes conexos; o Método da Caixa Limitante Adaptativa (COETZEE, 1998) (SOUZA, 2000) (CAMPOS, 2001), que emprega uma construção no formato de “L invertido” para, num primeiro momento encontrar os limites esquerdo e superior do caractere e após, os demais limites; e o Método da Etiquetação dos Componentes Conectados (NIJHUIS, 1995) (KWASNICKA, 2002) (BREMANANTH, 2005), que atribui uma etiqueta única aos *pixels* que compõem um mesmo objeto.

Ao final do processo, geralmente os caracteres encontrados são mapeados para uma forma de representação mais apropriada ao método de reconhecimento empregado antes de serem repassados para a próxima etapa.

2.5.7 Reconhecimento dos caracteres

Finalmente, cada um dos caracteres extraídos na etapa anterior é enviado para o módulo de reconhecimento, que disponibiliza como saída o código *American Standard Code for Information Interchange* (ASCII) do conjunto de dados recebido como entrada.

Segundo Polidório (2007) e Dias (2005), os principais métodos empregados no reconhecimento de caracteres são o Método Estatístico, que emprega uma medida de distância na procura da amostra de maior similaridade em um conjunto de amostras rotuladas; o Método Conexionista, o qual utiliza um conjunto de amostras rotuladas para treinar uma rede de elementos, tendo fronteiras de decisão não-lineares definidas iterativamente pelo método de treinamento e pelo conjunto de amostras; e o Método Sintático, que se baseia na estrutura dos padrões de entrada, onde uma gramática é empregada na definição das regras de formação de tais estruturas e um autômato utilizado na operacionalização da tarefa.

Questões como o reconhecimento parcial ou falso reconhecimento da placa de licenciamento veicular podem ou não ser aceitas, de acordo com a taxa de erro permitida pela aplicação (HI-TECH, 2007) (PHOTOCOP, 2007). No caso de controle de acesso, por exemplo, nenhuma falha do sistema é aceitável, pois isso pode acarretar na liberação do acesso a pessoas não autorizadas (falso positivo) ou negação ou acesso àquelas autorizadas (falso negativo). Já no caso de um sistema de estacionamento, o não reconhecimento de um caractere pode ser compensado pela análise de outras informações, como a existência de um registro de entrada cuja placa é similar à analisada no momento da saída.

Dependendo da forma como o sistema é implementado, outros blocos podem ser agregados. Muitas vezes é utilizada uma etapa de pré-processamento, que visa efetuar tarefas como conversão de sistema de cores ou algum aprimoramento na qualidade da imagem que melhore o desempenho nas etapas posteriores. Entretanto, tal procedimento pode ser executado em mais de um momento.

Outro exemplo que merece destaque é um bloco de “Enquadramento da placa”, citado por Dias (2007). Tal procedimento é executado após a etapa de localização da placa, visando limitar a região da mesma às suas bordas, bem como corrigir a distorção de perspectiva introduzida pela forma como a câmera é posicionada, conforme ilustrado na Figura 2.18, subitens (a),(b) e (c), respectivamente. O correto enquadramento da placa e a correção da distorção de perspectiva implicam diretamente em melhorias fundamentais na eficiência dos demais módulos. Obviamente o tempo de processamento é diretamente proporcional ao número de etapas executadas, bem como ao tipo de técnicas empregadas. Neste contexto o

desafio é justamente aumentar o percentual de reconhecimento do sistema sem que isso implique num aumento significativo no tempo de resposta do mesmo.

(a) (b) (c)

Figura 2.18 – Ilustração do processo de enquadramento da placa, proposto por Dias (2007).

2.5.8 Aplicações

Um SRPLV pode ser utilizado em toda e qualquer situação em que haja necessidade de identificação automática de veículos. Tal necessidade se aplica a uma gama enorme de aplicações, cuja tendência é de aumentar ainda mais. Dentre as já existentes, podem ser citadas (DIAS, 2007) (HI-TECH, 2007) (QUERCUS, 2007):

- Estacionamento: a placa do veículo é verificada tanto na entrada, quanto na saída do estabelecimento, permitindo o cálculo da taxa de serviço em função do tempo de permanência. Além da placa, o sistema também pode obter imagens dos ocupantes do veículo, visando inibir situações de roubo ou seqüestro;
- Controle de Acesso: similar ao controle exercido num estacionamento, mas o foco neste caso é permitir o acesso a áreas restritas apenas a membros autorizados, como no caso de uma área segura;
- Pedágio: a placa do veículo é identificada quando o veículo adentra a pista de pedágio, liberando imediatamente a cancela no caso de cobrança posterior ou agilizando o processo no caso de cobrança no ato;
- Controle de Fronteira: permite o automatizar o controle de entradas e saídas de um país, bem como agilizar tais procedimentos e verificar eventuais impedimentos;
- Identificação de veículo roubado: assim que ocorre o roubo ou furto de um veículo, a placa do mesmo pode ser incluída numa lista centralizada de

veículos nesta situação. A partir daí, o veículo pode ser localizado no momento em que for parado em um ponto de fiscalização ou for flagrado por alguma câmera de vigilância instalada em pontos estratégicos de uma cidade;

- Fiscalização: no caso do condutor do veículo cometer alguma infração de trânsito, como exceder o limite de velocidade ou avançar o sinal vermelho, a multa pode ser automaticamente apurada e enviada ao mesmo, a partir da placa do veículo conduzido por ele;
- Controle de tráfego: permite efetuar a triagem automática dos veículos, de acordo com suas permissões de entrada e direcioná-los, então, para vias distintas, reduzindo o congestionamento do tráfego;
- Ferramenta de *marketing*: as placas dos veículos podem ser utilizadas na criação de uma lista de visitantes freqüentes para propósitos de marketing, bem como na construção de perfis de tráfego (entradas por dia/hora, etc.);
- Controle de duração de viagem: a identificação de um veículo ao longo de um percurso permite apurar uma série de informações sobre o trajeto, como local e hora de origem e destino, velocidade média (que permite apurar também se a velocidade limite foi ultrapassada), acionamento imediato de uma equipe de socorro após um acidente ou parada não prevista, etc.

3 SISTEMAS EMBARCADOS

3.1 Introdução

Apesar do termo “microprocessador” estar intimamente ligado à figura do computador pessoal convencional, esta não constitui a única – tampouco a de maior demanda – área de aplicação do mesmo (HEATH, 2003). O surgimento do microprocessador ocorreu na década de 70, por conta da necessidade de flexibilização dos circuitos dedicados utilizados nas calculadoras da época. As primeiras calculadoras eletrônicas empregavam centenas de circuitos discretos e possibilitavam a execução das 4 operações básicas, apenas. A evolução da indústria de semicondutores permitiu que essa massa de circuitos fosse integrada em um único circuito dedicado. Essa integração permitiu a redução do custo de produção, mas ao mesmo tempo demandava o desenvolvimento de um novo circuito dedicado conforme novas funcionalidades fossem agregadas. A solução para este problema foi a criação de um circuito que permitisse alguma forma de programação, possibilitando a adição de novas funções sem que tais mudanças implicassem obrigatoriamente na alteração do mesmo. Nascia assim o microprocessador.

Atualmente, microprocessadores estão amplamente disseminados, embarcados em equipamentos dos mais diversos tipos, desde brinquedos infantis até aeronaves de combate. Nestes equipamentos, por sua vez, cada função que necessita de uma forma de controle eletrônico emprega um microprocessador, o qual se encontra inserido em um sistema embarcado. Desta forma, o propósito de um sistema embarcado é o de controlar algum aspecto de um sistema físico, como temperatura, movimento, etc., a partir de uma série de entradas. Com o advento da era digital e da conseqüente substituição das tecnologias analógicas no segmento dos equipamentos de consumo, a utilização de sistemas embarcados é cada vez maior.

3.2 Definições

Embora a literatura costume definir o termo *sistema embarcado* não pelo que ele significa, mas pelas diferenças em relação a um computador pessoal convencional, uma definição razoável para o mesmo é a de que se constitui num sistema micro-processado, de propósito específico, cuja interface com o usuário final se dá de forma indireta (HEATH,

2003) (COELHO, 1998) (BERGER, 2001). Apesar de consistir numa simplificação, tal definição é satisfatória, pois ilustra as principais características de um sistema embarcado. Dentre tais características, talvez a de maior relevância seja o fato de tais sistemas apresentarem uma funcionalidade restrita. Num sistema convencional (leia-se computador pessoal), o usuário final é quem define a funcionalidade de mesmo a partir da seleção do pacote de *software* adequado. Desta forma, hora o sistema pode se comportar como um editor de texto, hora como planilha eletrônica, e assim por diante. Em outras palavras, a funcionalidade do sistema pode ser re-programada pelo usuário final. Já num sistema embarcado, o usuário final pode até efetuar escolhas em relação à funcionalidade original, entretanto não tem a possibilidade de alterar a mesma.

No que tange a questão da interface com o usuário final, enquanto a mesma ocorre de forma direta no caso de um computador pessoal, o mesmo não acontece num sistema embarcado, onde a interface direta se dá com o ambiente ou com outro equipamento. Isto ocorre porque enquanto o computador pessoal consiste no sistema em si, um sistema embarcado é apenas um elemento de um sistema mais complexo, que, por sua vez, faz interface direta com o usuário final. Como exemplo claro de um sistema deste tipo, pode-se citar um automóvel. Dentre os vários sistemas embarcados existentes neste sistema, toma-se como exemplo o encarregado pelo sistema de frenagem anti-blocante. Neste caso, no momento da frenagem, a interface do usuário final ocorre com o veículo, que, por intermédio do pedal de freio, interage com o sistema embarcado responsável pelo controle de frenagem.

Além das questões que dizem respeito à funcionalidade e à forma de interface com o usuário final, outras diferenças entre sistemas convencionais e embarcados que se pode citar são as seguintes (BERGER, 2001):

- Sistemas embarcados são suportados por uma vasta gama de processadores e arquiteturas de processadores: diferentemente dos computadores pessoais convencionais, que na sua grande maioria são baseados num número restrito de plataformas (como a x86, da Intel, ou a SPARC, da Sun), sistemas embarcados empregam um número muito maior de plataformas em sua construção. Já no ano de 2000, havia mais de 140 tipos diferentes de processadores disponíveis no mercado, fabricados por cerca de 40 empresas distintas. Atualmente o número de fornecedores cresceu para algo em torno de 70, enquanto que a diversidade de dispositivos/núcleos gira em torno de 1000 itens (CRAVOTA, 2007);

- Sistemas embarcados são geralmente sensíveis ao custo: existem sistemas embarcados, como os utilizados em projetos espaciais, onde a área ocupada e a potência dissipada são fatores mais críticos do que o custo financeiro. Entretanto, em produtos de massa, como aqueles projetados para o mercado de consumo, o custo financeiro é tão ou mais importante quanto área e/ou potência dissipada. O custo do microprocessador em si é um fator, mas geralmente o que se visa é uma redução de custo no sistema como um todo;
- Sistemas embarcados têm restrições de tempo-real: o fato da operação de abertura de um documento demorar segundos num computador pessoal, por ocasião da carga inicial do *software* de processamento de texto, não tem influência alguma no resultado da operação. Entretanto, se tal atraso ocorrer no sistema embarcado de alerta de colisão de uma aeronave, as conseqüências podem ser catastróficas. Tal exemplo ilustra um sistema embarcado com *restrições críticas de tempo*, onde a tarefa é tida como falha se não executada dentro de uma janela de tempo. Já sistemas com *restrições sensíveis de tempo* são aqueles onde a violação da janela de tempo compromete, mas não invalida a função. Como exemplo, cita-se o controle de impressão de uma impressora jato de tinta, onde a violação da janela de tempo reduz o número de páginas impressas por minuto, sem que haja falha na impressão em si;
- Caso um sistema embarcado use um Sistema Operacional (SO), provavelmente é do tipo de tempo-real: *Real-Time Operating Systems* (RTOSs) permitem que as tarefas críticas sejam executadas dentro da janela de tempo necessária, independente do número de processos que estão rodando em um dado momento. Caso uma tarefa não seja concluída dentro da sua janela de tempo, a responsabilidade é do programador, não de outros fatores. Por conta desta disciplina rígida, RTOSs geralmente não apresentam as mesmas falhas apresentadas por SOs utilizados em computadores pessoais convencionais;
- As implicações das falhas de *software* são muito mais severas em sistemas embarcados do que em sistemas convencionais: muitos dos sistemas dos quais dependem vidas humanas, como os já citados sistema de frenagem de um veículo, sistema anti-colisão de uma aeronave, consistem em sistemas embarcados. Eventuais falhas de *software* em sistemas deste tipo podem comprometer tais vidas. Por conta desses aspectos, sistemas embarcados

tipicamente possuem mecanismos de auto-recuperação, como um *watchdog timer*, que restabelece o controle do dispositivo, no caso de haver a perda do mesmo por parte do *software*;

- Sistemas embarcados têm restrições de consumo de potência: muitos dos sistemas embarcados existentes, como os utilizados em sistemas de monitoramento de espécies selvagens, ou mesmo aqueles utilizados num automóvel, possuem como fonte de alimentação uma bateria. Deste modo, não apenas o microprocessador (que num sistema convencional pode consumir mais de 100 *Watts* de potência), mas o sistema como um todo deve ser projetado de modo a respeitar tais restrições de consumo;
- Sistemas embarcados devem operar sobre condições ambientais extremas: conforme mencionado no início deste capítulo, microprocessadores – e, por conseqüência, sistemas embarcados – estão amplamente disseminados. Seja sob temperaturas extremamente altas ou baixas, umidade abundante ou escassa, um sistema embarcado deve estar preparado para todas estas adversidades ambientais. Entretanto, a solução de um aspecto pode levar a implicações em outro, como no caso da proteção microprocessador mediante selagem, dificultando assim a dissipação de calor produzido pelo mesmo;
- Sistemas embarcados utilizam um conjunto restrito de recursos: enquanto sistemas convencionais dispõem de inúmeras interfaces de entrada e saída, bem como barramentos de diversos tipos, o conjunto de recursos disponíveis num sistema embarcado é muito mais restrito. Ao passo que este fato contribui para a redução de área, consumo de potência e custo financeiro do mesmo, também cria sérias limitações no tocante ao desenvolvimento deste;

3.2.1 Componentes de um sistema embarcado

Por conta da sua natureza restrita em vários aspectos, um sistema embarcado costuma ser composto pelo mínimo número possível de componentes. Mais do que isso, a capacidade de armazenamento/processamento e complexidade de tais componentes geralmente é a mínima necessária para satisfazer as especificações do sistema. Na seqüência são discutidos os principais elementos de um sistema embarcado citados por Heath (2003).

3.2.1.1 Processador

Assim como num sistema convencional, o microprocessador é o elemento principal de um sistema embarcado. É justamente pelo emprego deste elemento que se torna possível a agregação de novas funcionalidades num produto, de uma forma rápida e de baixo custo financeiro, considerando a implementação de tais funções em *software*.

O principal critério utilizado na seleção de um processador é a capacidade de processamento do mesmo. Entretanto, um dos aspectos que contribui para um desempenho final insatisfatório do sistema é o fato das tarefas executadas pelo processador selecionado serem freqüentemente sub-estimadas em relação ao seu tamanho e complexidade. A seleção do processador a partir de resultados obtidos pelo mesmo em *benchmarks* é outro fator que pode contribuir negativamente neste sentido, pois tais resultados podem não ser representativos em relação ao tipo de processamento exigido pelo sistema embarcado. Outros critérios utilizados na seleção do processador, os quais podem ser mais relevantes que o próprio poder de processamento, dependendo das restrições de projeto, são o custo financeiro, o consumo de potência, as ferramentas de *software* disponíveis para o mesmo, bem como a sua disponibilidade.

Dentre os tipos mais comuns de processadores, pode-se citar os seguintes:

- Microprocessador: processadores de propósito geral, cujo tamanho da palavra de dados pode variar de 8 a 64 *bits* e o conjunto de instruções ser complexo (CISC) ou reduzido (RISC). Podem ser compostos por um ou mais núcleos e apresentar versões próprias para implementação em dispositivos re-configuráveis (*soft processors)*;
- Microcontrolador: são na verdade sistemas autocontidos, compostos não apenas por um microprocessador, mas também por módulos *Random Access Memory* (RAM) e *Read-Only Memory* (ROM), gerador de *clock*, periféricos (*timer*, *watchdog*, conversor Analógico para Digital (A/D) e Digital para Analógico (D/A)) e dispositivos de entrada/saída (E/S) (geralmente interfaces de comunicação serial);
- *Digital Signal Processors* (DSPs): processadores otimizados para o processamento digital de sinais. Empregam recursos que maximizam o poder de processamento em aplicações deste tipo, como blocos multiplicador-acumulador (próprios para implementação de filtros digitais), deslocadores de barril e geralmente a arquitetura Harvard (que, em contraste com a

VonNeumann, possui memórias e caminhos distintos para programa e dados, o que permite acelerar a execução das instruções).

3.2.1.2 Memória

Os critérios de seleção do tipo e quantidade de memória são fortemente influenciados pelo projeto de *software*. Em contrapartida, a configuração de memória selecionada implica na forma como o *software* é projetado, escrito e desenvolvido.

Geralmente um sistema embarcado é composto tanto por módulos ROM, quanto por módulos RAM. Os módulos ROM, por serem não-voláteis, armazenam o código do programa executado pelo sistema. Quando não é possível o armazenamento do código completo, apenas uma rotina de inicialização (*bootstrap*) é armazenada no bloco ROM. Tal rotina é encarregada da carga do código completo, a partir de uma fonte externa, no momento da inicialização do sistema. Já os módulos RAM, os quais são do tipo volátil, possibilitam a execução do *software* disponibilizando armazenamento temporário para variáveis e estruturas utilizadas na execução das tarefas. Em função do maior custo financeiro de um módulo RAM em relação ao ROM, geralmente a disponibilidade do primeiro é reduzida, o que implica na escrita de um *software* que minimize a utilização deste recurso.

3.2.1.3 Periféricos

Um sistema embarcado utiliza periféricos para se comunicar com o mundo externo. Os periféricos de entrada geralmente consistem em sensores que medem algum aspecto do ambiente externo (como temperatura, pressão, altitude, etc.) e assim determinam de forma efetiva as saídas disponibilizadas pelo sistema embarcado. Dentre os principais tipos de periféricos, se pode citar os seguintes:

- Binários: pinos externos simples, cujo estado lógico pode ser 1 ou 0. Podem ser utilizados individualmente ou agrupados, formando interfaces paralelas;
- Seriais: interface composta de um ou dois pinos, que emprega o modo serial de comunicação. Facilita a conexão ao passo que dificulta a programação;
- Analógicos: quando a interface é realizada diretamente com o mundo externo, faz-se necessário o uso de conversores A/D e D/A, já que o sistema em si opera apenas no domínio digital;

- *Displays*: constituem uma forma de externalizar informações geradas pelo sistema. Como exemplos se pode citar *Ligth Emission Diodes* (LEDs), *displays* de sete segmentos e painéis *Liquid Crystal Display* (LCD);
- Saídas derivadas de tempo: temporizadores e contadores são provavelmente as funções mais utilizadas num sistema embarcado.

3.2.1.4 *Software*

Mais do que apenas o programa que é executado pelo sistema, o elemento *software* também abrange a tecnologia que agrega valor ao sistema e define o desempenho que o mesmo é capaz de obter. Neste contexto, outros componentes que se pode citar são:

- Inicialização e configuração;
- Sistema Operacional ou ambiente de execução;
- Tratamento de exceções;
- Suporte a depuração de código e manutenção.

3.2.1.5 Algoritmos

Os algoritmos se constituem no componente principal do elemento de *software* e definem as funcionalidades disponibilizadas por um sistema embarcado, bem como a forma que elas são implementadas. A escolha dos algoritmos constitui um ponto crítico, pois conforme já mencionado, a idéia de satisfazer os requisitos do sistema, empregando para isso o mínimo de recursos necessários, geralmente é premissa de um sistema embarcado. A utilização de algoritmos otimizados permite a execução de uma mesma tarefa numa plataforma mais enxuta, o que permite a redução não apenas do custo financeiro, mas também da área e consumo de potência do mesmo.

3.2.2 Aplicações

Além das aplicações já citadas nas seções anteriores, sistemas embarcados também podem ser empregados em inúmeras outras áreas, como (EMBEDDED, 2007) (DE MICHELI, 1994) (CHIODO, 1994) (KUTTNER, 1996) (FONS, 2006):

- Telecomunicações: de centrais telefônicas a telefones móveis, incluindo também equipamentos de rede computacional, como roteadores e *bridges*;
- Eletrônica de consumo: *Personal Digital Assistants* (PDAs), aparelhos de *MPEG-1 Audio Layer 3* (MP3), Compact Disc (CD) e Digital Vídeo Disc (DVD), consoles de *vídeo-game*, receptores de *Global Positioning System* (GPS), etc.;
- Eletrodomésticos: fornos de microondas, lavadoras de louça e de roupas, refrigeradores, televisores, condicionadores de ar, etc.;
- Transportes: desde sistemas terrestres (controle de estabilidade, tração, combustão, etc.) até aéreos (sistema de direção inercial, comunicação, navegação, metereológico, etc.);
- Medicina: monitoramento de sinais vitais, estetoscópios eletrônicos, diagnóstico por imagens (tomografia, ressonância magnética), etc.;
- Militar: radar, sonar, reconhecimento automático de alvo, guerra eletrônica (manipulação do espectro eletromagnético, impossibilitando o uso do mesmo pelos inimigos e otimizando para os aliados), inteligência de sinais (interceptação e decodificação de mensagens), etc.;
- Outros usos: reconhecimento de impressões digitais, inspeção de colheita, monitoramento marinho (rastreamento de cardumes e mamíferos marinhos, identificação de derramamento de óleo), etc..

3.3 Co-projeto de *hardware* e *software*

3.3.1 Definições

De uma maneira simplificada, o co-projeto de *hardware* e *software* pode ser definido como o desenvolvimento concorrente de *hardware* e *software*, a partir de uma metodologia comum (ADAMS, 1996). Tal metodologia é empregada no desenvolvimento de sistemas heterogêneos, dentre os quais se pode citar (NIEMANN, 2007) (ADAMS, 1996):

- Sistemas embarcados;
- Sistemas multiprocessados heterogêneos;
- Processadores de conjunto de instruções de aplicação específica;
- Unidades funcionais de propósito específico;

- Co-processadores de aplicação específica.

Tradicionalmente, o fluxo de projeto baseado na metodologia de co-projeto de *hardware* e *software* é dividido, basicamente, em três etapas, conforme ilustrado na Figura 3.1. A primeira etapa se constitui no levantamento dos requisitos do sistema, tanto os funcionais, ou seja, o que exatamente o sistema deve fazer, quanto os não-funcionais, como velocidade, atraso, custo e consumo de potência (COELHO, 1998). Inicialmente tais requisitos são apurados informalmente, com o uso de linguagem natural Em seguida, são organizados de uma forma mais estruturada, de uma maneira que considerações técnicas possam ser efetuadas sobre os mesmo. Com base nestas considerações é que se dá o particionamento dos recursos, o qual consiste na especificação de quais partes do sistema serão implementadas em *software*, e quais serão compostas por *hardware*. Neste contexto, os requisitos não-funcionais exercem uma grande influência, pois a redução no custo final do produto geralmente está associada ao mapeamento do maior número possível de funções para *software*, enquanto que a implementação de tarefas em *hardware* propicia uma aceleração na execução e redução no consumo de potência das mesmas. Ainda nesta etapa são definidos aspectos que devem ser comuns às partes de *hardware* e de *software*, como a forma de representação dos dados (*big-endian* ou *little-endian*), protocolos de comunicação, etc.

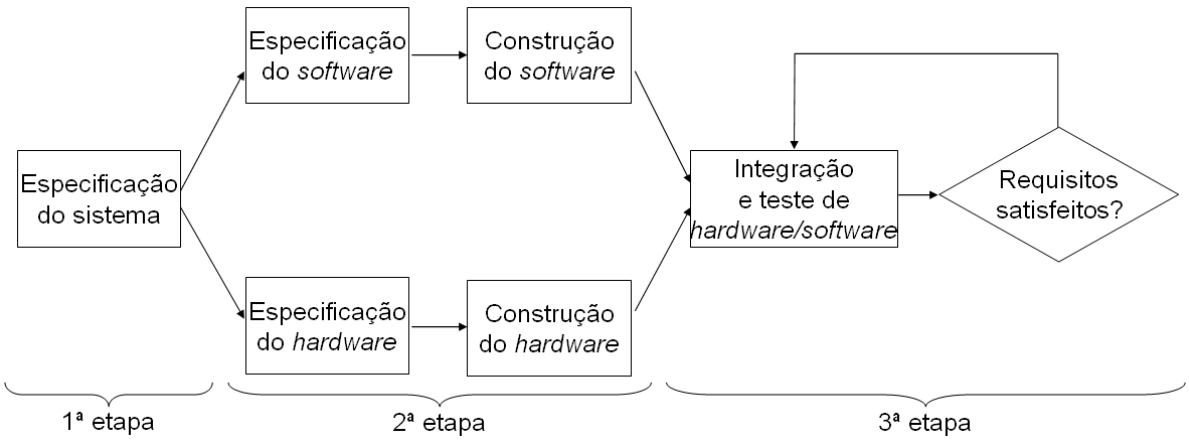


Figura 3.1 – Fluxo tradicional de co-projeto de *hardware* e *software*.

Na segunda etapa o fluxo se divide em dois caminhos distintos, mas que seguem de forma paralela: especificação e construção de *hardware* e especificação e construção de *software*. Após o desenvolvimento da infra-estrutura de *hardware*, a mesma é validada de forma individual pela escrita de *software* básico de teste, que verifica, por exemplo, as interfaces com a memória, entre os chips e com processador. Enquanto isso, o *software* que

irá ser executado sobre essa plataforma de *hardware* é implementado. Na falta da plataforma real para a execução dos testes, geralmente um emulador é construído, permitindo assim que testes básicos de interface com o Sistema Operacional sejam efetuados, bem como possibilitando a otimização e validação dos algoritmos (TOWNSEND, 2005).

Finalmente, na terceira fase ocorre a integração dos caminhos que haviam se dividido logo após a especificação inicial. O *software* é então integrado ao *hardware* e os testes de integração são executados. Os requisitos iniciais são então verificados e o projeto é tido como finalizado caso os mesmos tiverem sido satisfeitos. Caso contrário, o processo entra num laço iterativo de ajustes de integração seguidos por verificação de requisitos até que os mesmos sejam satisfeitos. Nesta etapa, os ajuste dizem respeito principalmente, senão totalmente, às partes de *software*, já que o re-projeto das partes de *hardware* é demorado e custoso, de um ponto de vista financeiro.

3.3.2 Limitações

A partir da descrição do fluxo tradicional dada na seção anterior, é possível antever uma série de dificuldades, sendo que as principais se originam nas desconexões no particionamento de *hardware* e *software* que são detectadas apenas durante a etapa de integração. Tais incompatibilidades descobertas de forma tardia incorrem não apenas na extrapolação do orçamento e do prazo do projeto, mas também na possibilidade do sistema não satisfazer os requisitos iniciais, no pior dos casos (KUTTNER, 1996).

A causa de tais problemas está relacionada à utilização do modelo de desenvolvimento em cascata, onde as tarefas são executadas de forma isolada e desconexa, tornando assim o processo difícil, custoso e altamente susceptível a erros. A maior prova desta ineficiência do modelo reside justamente no fato das incompatibilidades entre *software* e *hardware* serem endereçadas apenas no final do processo (KUTTNER, 1996).

3.3.3 Evolução

Diante das incontáveis dificuldades apresentadas pelo modelo tradicional de co-projeto de *hardware* e *software*, novas metodologias, cujo principal intuito era o de explorar a heterogeneidade deste tipo de sistemas, começaram a ser propostas. Durante a década de 90 os esforços se concentraram na tentativa do desenvolvimento de um ambiente que favorecesse

o aspecto cooperativo do processo e que possibilitasse a modelagem e simulação do sistema como um todo.

Segundo De Micheli (1994) e Chiodo (1994), o suporte das ferramentas de *Computer Aided Design* (CAD) da época ao co-projeto de *hardware* e *software* era ainda primitivo, pois os modelos não eram abstratos o suficiente para permitir uma modelagem em nível de sistema. Em contrapartida, tais modelos também não possuíam detalhamento suficiente que possibilitasse um particionamento eficiente de recursos a partir de uma estimativa de custos (neste caso, área, consumo de potência, velocidade, etc.) em nível físico.

Entretanto, a percepção de tais carências abriu espaço para a pesquisa e desenvolvimento destas áreas, dando origem a um modelo mais robusto, capaz de endereçar as fraquezas do anterior, como o ilustrado na Figura 3.2. Num ambiente baseado neste

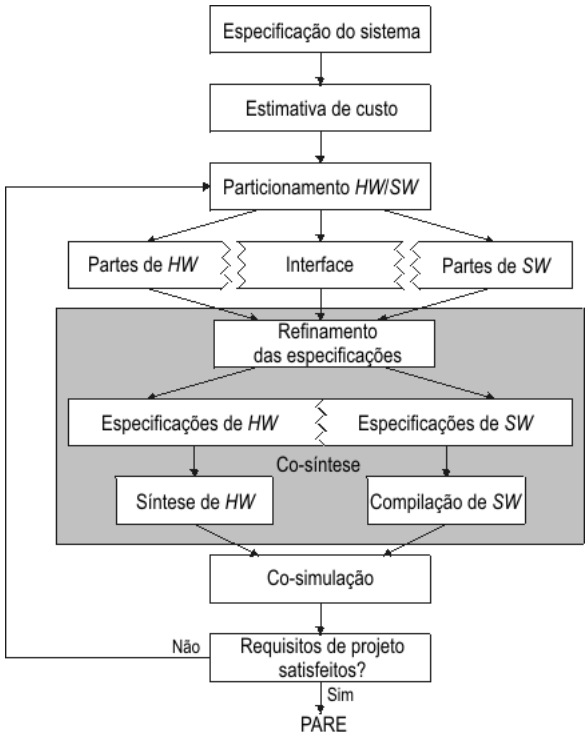


Figura 3.2 – Fluxo atual de co-projeto de *hardware* e *software.*

modelo, as especificações em nível de sistemas são inicialmente fragmentadas em blocos básicos, denominados grânulos. O particionamento é efetuado a partir de um mapeamento ótimo dos grânulos, o qual utiliza como parâmetro uma estimativa de custo gerada a partir das métricas de custo determinadas para cada um dos grânulos. Como métricas de custo entende-se área, tempo de execução, consumo de potência, etc. no caso de *hardware* e tempo de execução e consumo de memória (de programa e dados) no caso de *software*. Em seguida

ocorre a co-síntese das especificações, pelo mapeamento dos grânulos para suas respectivas implementações lógicas de *hardware* e *software* e posterior síntese dos elementos de *hardware* e compilação dos elementos de *software*. O desempenho final do sistema é então obtido a partir da co-simulação do conjunto de *Application Specific Integrated Circuits* (ASICs) e programas *assembler* resultante da etapa anterior. Neste momento, caso as restrições de desempenho e custo forem satisfatórias, o processo é encerrado. Caso contrário, o ciclo e reiniciado a partir da etapa de particionamento e repetido até que restrições aceitáveis sejam obtidas.

Como exemplo de sistemas que disponibilizam conjuntos de ferramentas integradas para o projeto automatizado de sistemas podem ser citados os ambientes METROPOLIS e SONORA (GUPTA, 2002). METROPOLIS é um ambiente de desenvolvimento para sistemas de tempo-real, que decompõe uma especificação num conjunto de processos concorrentes e intercomunicantes. O particionamento é feito de forma manual. Já o ambiente SONORA tem como objetivo o desenvolvimento de sistemas embarcados, utilizando para isto uma combinação de ferramentas comerciais e acadêmicas.

3.3.4 Tendências

Uma das tecnologias que tem contribuído para avanços significativos na área de co-projeto de *hardware* e *software* é a computação re-configurável (WOLF, 2003). Historicamente, FPGAs têm sido usados como forma de conectar os vários elementos de um sistema (DAVIS, 2005), como sistemas de emulação para ASICs (NIEMANN, 2007), ou como co-processadores re-configuráveis (NARASIMHAN, 1996).

Entretanto, as gerações mais recentes de FPGAs tem disponibilizado não apenas cada vez mais área, como também mais recursos embutidos no próprio dispositivo, fazendo com que o mesmo deixe de ser apenas um elemento e adquira o *status* de plataforma de desenvolvimento. Como exemplos de recursos dedicados que têm sido acrescentados nos FPGAs podem ser citados blocos multiplicadores-acumuladores, blocos RAM, manipuladores de *clock*, *transceivers* e inclusive microprocessadores (XILINX, 2007).

Concomitantemente com a evolução dos FPGAs foram desenvolvidos métodos de desenvolvimento de sistemas embarcados baseados em plataforma FPGA, como o proposto por Davis (2005). Tal método consiste num fluxo centrado em *software*, ou seja, que utiliza uma linguagem de alto nível, como C/C++, para a concepção do sistema. O particionamento é efetuado automaticamente com o uso de compiladores *C-to-hardware*, a partir das métricas de

desempenho obtidas pelo perfilamento da aplicação. A Figura 3.3 ilustra uma arquitetura projetada com base neste fluxo, onde o microprocessador utilizado pode ser tanto um *hard processor* (módulo físico, embutido no FPGA) ou *soft processor* (módulo lógico, implementado na área programável do FPGA).

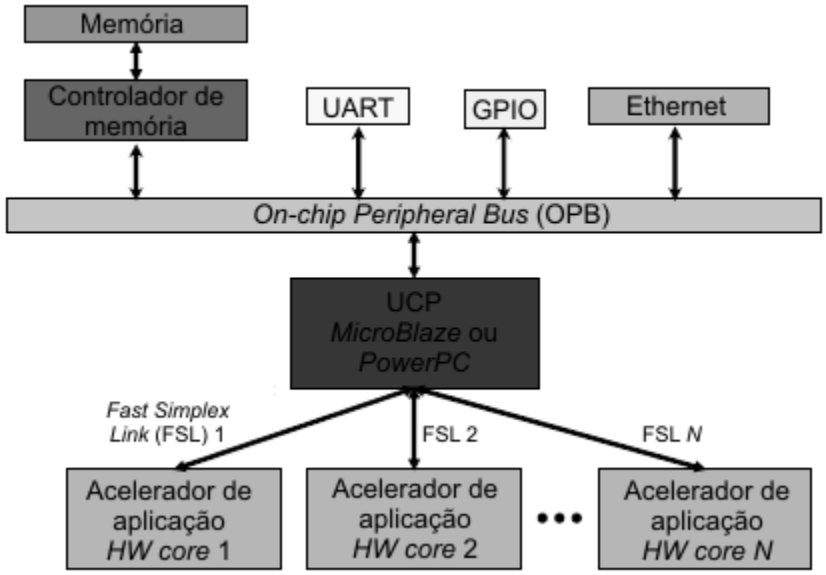


Figura 3.3 – Arquitetura co-processada com múltiplos aceleradores em *hardware.*

4 SISTEMA EMBARCADO PARA EXTRAÇÃO E RECONHECIMENTO DE CARACTERES ÓPTICOS

4.1 Introdução

Conforme discutido brevemente no capítulo introdutório, a proposta deste trabalho consiste no desenvolvimento de uma solução embarcada capaz de resolver a problemática da extração e reconhecimento de caracteres ópticos presentes em imagens digitais binárias. A partir das considerações realizadas no Capítulo 3, a plataforma adotada na construção de tal sistema embarcado consiste num dispositivo FPGA. O dispositivo utilizado neste trabalho (Xilinx XC2VP30-FF896) possui dois microprocessadores *PowerPC 405* fisicamente implementados (*hard cores*) no seu interior, permitindo assim que se possa optar pelo emprego de *hard* ou *soft cores*.

A escolha de um modelo heterogêneo para a construção do sistema se justifica pelas vantagens apresentadas pelo mesmo em relação a modelos homogêneos. Num modelo heterogêneo, os blocos construtores do sistema podem ser implementados tanto em *hardware*, como em *software*, de acordo com as restrições existentes, enquanto no modelo homogêneo os blocos são implementados por um elemento apenas (*hardware* ou *software*). Desta maneira, o emprego de um modelo heterogêneo possibilita estabelecer um compromisso entre a flexibilidade de um sistema totalmente em *software* (aplicativo) e o alto desempenho de um sistema totalmente em *hardware* (ASIC). Em outras palavras, para um dado sistema, a versão embarcada possui um maior desempenho, menor área e consumo de potência que a versão totalmente em *software*, ao mesmo tempo em que apresenta uma redução de custos financeiros, complexidade e *time to market* em relação à versão exclusivamente em *hardware*.

No que diz respeito à escolha da plataforma FPGA, vale ressaltar que a mesma se deu principalmente em função de duas características apresentadas pelo dispositivo: flexibilidade e praticidade. Enquanto sistemas embarcados baseados em micro-controladores são práticos no sentido de permitir a implementação do sistema em um *chip* único, não são flexíveis no que diz respeito à possibilidade de aumentar o desempenho a partir do uso de *hardware* dedicado. Já no caso de DSPs, ocorre o inverso. Com o advento de microprocessadores embarcados, FPGAs permitem tanto a aceleração de funções via *hardware* dedicado, como a implementação do sistema em um *chip* único (salvo o caso de interface direta com o mundo externo, onde conversores A/D e D/A são necessários). No caso desta proposta, o sistema

embarcado é composto pelo FPGA e um *chip* de memória volátil externa, este último utilizado na execução do *software* embarcado.

De acordo com a informação também já mencionada no capítulo inicial, o sistema implementado neste trabalho utilizou um SRPLV como estudo de caso. No intuito de contextualizar SRPLVs em relação à classificação de três níveis discutida no Capítulo 2, tem-se a Figura 4.1. Nesta figura se verifica que dois dos três principais módulos componentes de um SRPLV são classificados como de nível intermediário no contexto de visão computacional, pois utilizam métodos de segmentação (binarização, geralmente), bem como de representação e descrição (etiquetação de componentes conectados, códigos de cadeia, *skeletons*, etc.). Já o outro módulo principal (reconhecimento de caracteres) realiza processamento de alto nível, haja vista que emprega métodos de reconhecimento de padrões, como casamento de padrões e RNAs. Eventualmente, uma mesma técnica pode ser utilizada em níveis diferentes, como no caso da binarização, que pode ser empregada tanto como técnica de pré-processamento, quanto como técnica de segmentação.

Figura 4.1 – Relação entre os principais módulos de um SRPLV e a classificação de três níveis adotada na área de visão computacional.

Na próxima seção a proposta é discutida de forma detalhada, sem no entanto mencionar as questões referentes à implementação da mesma. Tais questões serão abordadas no capítulo seguinte.

4.2 A proposta

Não por acaso os blocos referentes à extração e reconhecimento de caracteres foram destacados na Figura 4.1. Esse artifício foi propositalmente empregado no intuito de frisar que

o foco desta proposta se concentra em tais blocos, enquanto a localização da placa foi alvo de pesquisa de Pacheco (2007), num trabalho conduzido praticamente em paralelo a este.

Assim sendo, a Figura 4.2 apresenta o diagrama de blocos do sistema proposto. A entrada é dada por uma imagem binária na qual espera-se estar contida a placa de licenciamento de um veículo. Já a saída consiste nos caracteres que formam a mesma, representados no formato ASCII, caso haja realmente alguma placa na imagem. Essa imagem de entrada é obtida a partir das coordenadas fornecidas pelo módulo implementado por Pacheco (2007). Tais coordenadas identificam uma região retangular na imagem de um veículo em trânsito, onde existe uma alta probabilidade de estar situada a placa de licenciamento do mesmo. Já em relação à saída do sistema, a mesma é composta por sete caracteres alfanuméricos, sendo os três primeiros constituídos por letras e os quatro últimos por números, haja vista que esta proposta aborda placas de licenciamento veicular no padrão brasileiro.

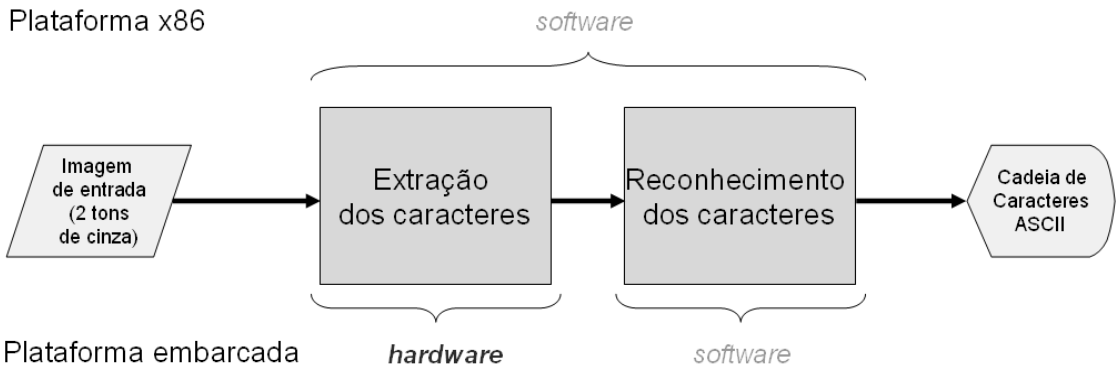


Figura 4.2 – Digrama de blocos da proposta desenvolvida neste trabalho.

Geralmente, como é o caso de Pacheco (2007), a etapa de localização da placa de licenciamento veicular é efetuada a partir de uma imagem em 256 tons de cinza, sendo a mesma convertida para dois tons de cinza na etapa de extração dos caracteres. Entretanto, como esta proposta leva em consideração uma imagem de entrada em dois tons de cinza, será necessária a implementação de um processo de conversão no intuito de contornar essa questão. Tal processo, denominado segmentação, será implementado como parte do sistema sobre plataforma x86, mas não comporá, entretanto, o sistema de extração e reconhecimento aqui proposto. Dentre as técnicas de segmentação, a mais comum é a denominada limiarização de níveis de cinza (ou apenas limiarização, ou também binarização) (SONKA, 2007). Tal técnica é extensivamente utilizada em SRPLVs, tanto pelo emprego do método de limiar global, quanto pelo de limiar local (LOTUFO, 1990) (NIJHUIS, 1995) (BARROSO,

1997) (SOUZA, 2000) (DIAS, 2005). A Figura 4.3 ilustra o diagrama do processo auxiliar de geração da imagem de entrada para o sistema de extração e reconhecimento de caracteres.

Como entrada deste processo, têm-se as coordenadas da região candidata, dadas pelo módulo desenvolvido por Pacheco (2007), bem como a imagem original em 256 tons de cinza utilizada na obtenção das mesmas. A região candidata é então extraída da imagem original e em seguida segmentada pela aplicação do processo de limiarização, dando origem, assim, à imagem em dois tons de cinza. No intuito de selecionar o método de limiarização capaz de gerar as entradas que melhor se adequam ao sistema proposto, tanto um método de limiar global (Otsu), quanto um de limiar local (Niblack) serão avaliados.

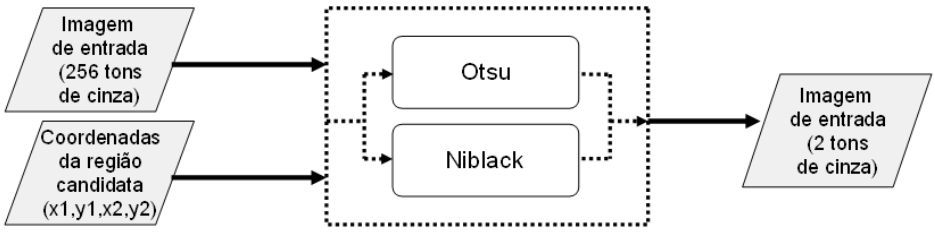


Figura 4.3 – Diagrama de blocos do processo auxiliar utilizado na geração das entradas do sistema aqui proposto.

Conforme já mencionado na Seção 1.3, uma versão em *software* do sistema foi inicialmente construída sobre plataforma x86, no intuito tanto de avaliar uma série de métodos selecionados, quanto de elaborar um parâmetro de comparação para a versão heterogênea do sistema construída na seqüência. No tocante à natureza dos métodos selecionados para avaliação, todos eles estão relacionados à etapa de extração de caracteres, haja vista que o emprego de RNAs na etapa de reconhecimento já havia sido definido desde o primeiro esboço da proposta. Os motivos que conduziram a esta decisão são discutidos mais adiante, na Seção 4.2.2.

Outra definição tomada *a priori* diz respeito ao particionamento dos elementos de *hardware* e *software* do sistema embarcado. Conforme ilustrado na Figura 4.2, definiu-se que as tarefas relacionadas ao módulo de extração de caracteres teriam uma implementação dedicada em *hardware*, enquanto o processo de reconhecimento dos caracteres seria efetuado em *software*, com o auxílio de um microprocessador (*hard* ou *soft*) embarcado no FPGA. Essa decisão foi baseada nas seguintes heurísticas:

- muitas das tarefas executadas no módulo de extração de caracteres basicamente se resumem no processamento seqüencial de uma matriz de *pixels*, adotando

operações de baixo custo de *hardware*, como comparações ou acumulações. Assim sendo, o desempenho das mesmas pode ser significativamente aumentado pela paralelização de tais operações, ao mesmo tempo em que os requisitos de área e de complexidade são mantidos em níveis aceitáveis;

- apesar de RNAs apresentarem uma estruturação propícia para implementação em *hardware*, o custo de área de uma rede totalmente conectada tende a ser elevado, já que normalmente cada neurônio requer um multiplicador distinto (levando em consideração que a fase de treinamento não necessita ser efetuada no sistema).

A seguir, cada um dos blocos componentes do sistema é discutido de uma forma mais detalhada.

4.2.1 Extração dos caracteres

A primeira etapa executada pelo sistema é a extração dos caracteres contidos na imagem de entrada, etapa esta por vezes denominada segmentação dos caracteres. Conforme ilustrado no diagrama de blocos da Figura 4.4, o módulo é composto por duas tarefas principais: a localização e a normalização de objetos. A idéia geral é a de identificar e separar os objetos presentes em uma imagem, redimensionando posteriormente aqueles cujas dimensões estejam dentro de uma faixa pré-estabelecida.

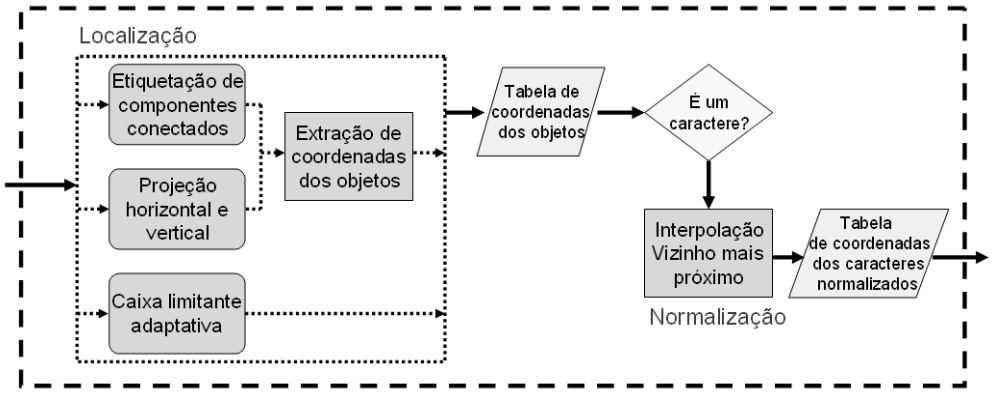


Figura 4.4 – Fluxograma referente ao bloco de extração dos caracteres, anterior à análise dos métodos.

Tendo como entrada uma imagem binária, onde os *pixels* de cor preta identificam os objetos e os de cor branca, o fundo, a tarefa de localização de objetos é responsável pela geração de uma tabela de coordenadas dos objetos. Cada registro existente nesta tabela

identifica um objeto distinto e fornece as coordenadas superior esquerda e inferior direita do retângulo virtual que define os limites do objeto.

Dentre os métodos mais comumente utilizados na realização da tarefa de localização dos caracteres encontram-se a etiquetação de componentes conectados, a projeção horizontal e vertical e a caixa limitante adaptativa. No caso dos dois primeiros métodos, as coordenadas das regiões encontradas devem ser apuradas por intermédio de um processo adicional, haja vista que a localização das mesmas é dada de forma indireta, diferentemente do último método. No intuito de avaliar os prós e contras da utilização, bem como o desempenho de cada um deles, todos foram selecionados para a implementação inicial em *software*.

Na seqüência do processo de extração dos caracteres é executada a tarefa de normalização dos mesmos, a partir da tabela que contém as coordenadas dos objetos localizados na imagem. Essa tabela é processada de forma seqüencial, no intuito de executar a normalização apenas dos objetos classificados como caracteres. Essa classificação se faz necessária para evitar que outros elementos não-alfanuméricos que porventura venham a compor a imagem (bordas da placa, parafusos de fixação, sujeira, etc.) sejam disponibilizados como saída do módulo.

Uma das formas utilizadas para classificar os objetos consiste na verificação empregada ao final do método da caixa limitante adaptativa, que se baseia nas dimensões do objeto em questão. Caso as dimensões do objeto estejam dentro de uma faixa pré-estabelecida, o mesmo é considerado um caractere, caso contrário, não. Neste sentido, os parâmetros utilizados pelo critério (altura e largura) são definidos a partir da distância entre a região de interesse (a placa, neste caso) e a câmera no momento da aquisição da imagem. Desta forma, quanto maior a distância, menor a região e vice-versa.

Cada objeto classificado como caractere é então normalizado, no intuito de descrever a região em questão de uma forma apropriada para a próxima etapa de reconhecimento, conforme descrito anteriormente na Seção 2.2. Neste sentido, a normalização consiste no redimensionamento do caractere para uma matriz de tamanho fixo, cujas dimensões são definidas e justificadas em seguida, na seção referente ao módulo de reconhecimento de caracteres. Como premissa, o método a ser utilizado no processo de redimensionamento deve permitir ambas as operações de ampliação e redução.

Após uma revisão bibliográfica sobre redimensionamento de imagens, que incluiu referências como (GONZALES, 2002) (KIM, 2003) (ANDREADIS, 2005) e (HWANG, 1997), optou-se pelo emprego da variante mais simples deste método: a interpolação “vizinho mais próximo”. A justificativa para tal escolha reside no fato de que a imagem produzida

como saída deste módulo não se destina à audiência humana, mas sim a um processo computacional de reconhecimento de padrões. Neste caso, mais importante que a fidelidade visual entre a imagem original e a redimensionada é a similaridade entre a segunda e a sua respectiva classe. Uma vez que um mesmo método de redimensionamento é utilizado para gerar o conjunto de entradas utilizadas no treinamento do classificador e as entradas do sistema em produção, a fidelidade visual é uma característica irrelevante.

Na seqüência, é dada a teoria sobre os métodos discutidos nesta seção.

4.2.1.1 Etiquetação de Componentes Conectados

Conforme mencionado na Seção 2.2, ao término do processo de segmentação de uma imagem, as regiões encontradas necessitam ser identificadas para que a descrição das mesmas seja possível. Tal processo é denominado *identificação de regiões* e uma das formas de realização do mesmo consiste no emprego da técnica de *etiquetação de componentes conectados*, também chamada *coloração* ou *etiquetação*, apenas (SONKA, 2007). Essa técnica se baseia nos conceitos de *vizinhança* (Seção 2.3.3.1) e *conectividade* (Seção 2.3.3.2) dos *pixels* para atribuir uma etiqueta única para cada região distinta existente numa imagem. Assumindo que uma imagem segmentada R é composta por m regiões distintas R_i, as quais englobam o fundo da imagem e os objetos encontrados na mesma, o conjunto que identifica estes últimos é dado por

$$R_b^C = \bigcup_{i=1, i \neq b}^{m} R_i$$

onde R^C é o complemento do conjunto, R_b é o fundo da imagem e as demais regiões são consideradas como objetos.

A entrada de um algoritmo de etiquetação consiste numa imagem mono- (binária) ou multi-níveis . Já a saída se constitui numa imagem por vezes denominada simbólica, onde o fundo é identificado por etiquetas de valor zero, enquanto os objetos por etiquetas de valor maior que zero. No caso de uma imagem de entrada binária, os *pixels* brancos denotam o fundo, enquanto os pretos, os objetos. Entretanto, nada impede que seja usada a lógica inversa, conforme ilustrado na Figura 4.5(a).

O processo de etiquetação é composto por duas etapas: atribuição de etiquetas e fusão de etiquetas. A primeira etapa consiste na atribuição de uma etiqueta inicial a cada *pixel* que compõe um objeto. Entretanto, durante este processo, etiquetas de valor diferente podem ser atribuídas à *pixels* que pertencem a um mesmo objeto, conforme ilustrado na Figura 4.5(b).

Desta forma, cada vez que uma situação destas ocorre, um par equivalente é gerado e armazenado para posterior processamento. Tal processamento, ilustrado na Figura 4.5(c), ocorre justamente na segunda etapa, onde a etiqueta previamente atribuída a cada *pixel* é verificada na tabela de equivalências e substituída, caso necessário (leia-se: caso divergirem).

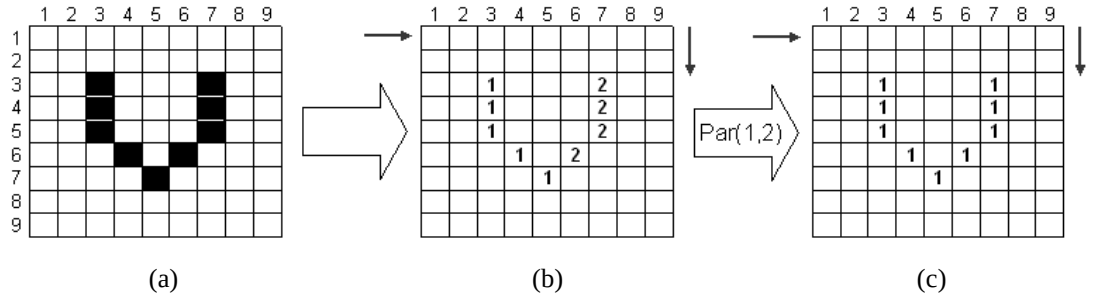


Figura 4.5 – Ilustração do processo de Etiquetação de Componentes Conectados.

Diversos métodos seqüenciais (ROSENFELD, 1966) (JEAN, 1994) (NICOL, 1995) e paralelos (RASQUINHA, 1997) (WANG, 2003) (YANG, 2005) foram propostos para resolver o problema de componentes conectados. Dentre os métodos seqüências, Rosenfeld (1966) propôs o algoritmo clássico, o qual efetua uma varredura dupla na imagem de entrada, da direita para a esquerda, de cima para baixo, armazenando os pares equivalentes numa ampla tabela. Já dentre os métodos pararelos, Yang (2005) propôs aquele considerado aqui como estado da arte, o qual emprega dois elementos de processamento e armazena os pares equivalentes numa matriz de classes. Como resultado, o método proposto por Yang (2005) obtém uma redução tanto de espaço para a alocação dos pares equivalentes, como no tempo de processamento da imagem.

No tocante ao funcionamento do algoritmo proposto por Yang (2005), o mesmo utiliza uma janela de três linhas por quatro colunas, como a ilustrada na Figura 4.6, que é deslocada sobre a imagem de entrada, da esquerda para a direita, de cima para baixo, processando os *pixels* P1 e P2 simultaneamente. Considerando os *pixels* L1~L6 como previamente processados, inicialmente o valor de P1 é verificado. Caso seja igual a zero (fundo), nenhum par equivalente é gerado, caso contrário o valor da etiqueta do mesmo é apurado a partir da sua vizinhança, pela seguinte ordem de prioridade: L3, L5, L1, L2. Quando nenhum dos vizinhos possuir uma etiqueta maior que zero, uma nova é gerada. Neste momento, um par equivalente é gerado caso as etiquetas de (L3,L5) ou (L3,L1) sejam diferentes, desde que também maiores que zero. Cada par equivalente endereça duas posições na matriz de classes. Quando um par é gerado, os conteúdos das posições endereçadas são comparados e o maior

valor é substituído pelo menor. Um exemplo deste processo é exibido na Figura 4.7, onde inicialmente cada elemento possui um conteúdo distinto (Figura 4.7(a)). A matriz de classes vai então se transformando, conforme os pares são adicionados: primeiramente o par (1,3) (Figura 4.7(b)), então o par (2,4) (Figura 4.7(c)) e finalmente o par (1,4) (Figura 4.7(d)). Concomitantemente, o mesmo processamento aplicado à P1, é aplicado à P2, mas neste caso considerando a vizinhança P1,L6,L4,L5. Na segunda etapa, conforme mencionado anteriormente, a imagem simbólica resultante é percorrida por P1 e P2, sendo os valores das etiquetas alterados conforme o conteúdo da matriz de classes.

	L1	L2	L3
L4	L5	P1	
L6	P2		

Figura 4.6 – Janela de processamento proposta por Yang (2005).

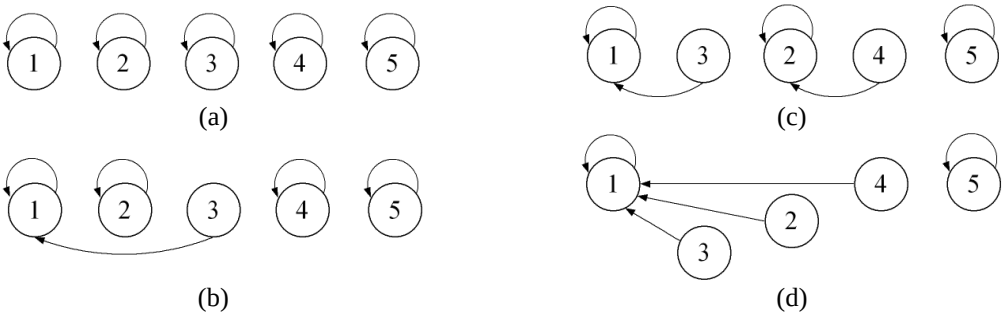


Figura 4.7 – Ilustração das iterações que ocorrem na matriz de classes.

4.2.1.2 Projeção horizontal e vertical

A projeção, ou projeção de amplitude, é um método que basicamente consiste no somatório unidimensional do número de *pixels* que possui um determinado nível de cinza. Neste sentido, os dois tipos mais comuns de projeção são o horizontal e o vertical, dados, respectivamente, pelas seguintes equações

$$H(y) = \sum_{x=1}^{X} f(x, y) \quad \textbf{(4.1)}$$

$$V(x) = \sum_{y=1}^{Y} f(x, y) \quad \textbf{(4.2)}$$

onde *f(x,y)* é uma imagem digital, *X* o número de colunas desta imagem e *Y* o número de linhas.

Tomando como exemplo uma imagem representada em dois níveis de cinza (binária), onde o somatório considera o número de *pixels* pretos em uma mesma direção, o resultado das projeções horizontal e vertical é ilustrado na Figura 4.8. A partir dos resultados das projeções é possível extrair os caracteres com base na técnica inicialmente proposta por Lu (1980 *apud* BARROSO, 1997), denominada picos e vales, que define as fronteiras dos objetos a partir dos vales existentes nas projeções.

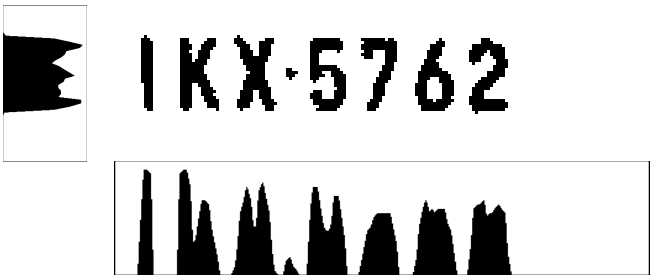


Figura 4.8 – Resultado obtido pelo método de projeções.

4.2.1.3 Caixa Limitante Adaptativa

Outro método também utilizado na identificação das regiões existentes numa imagem segmentada é o citado por Souza (2000), proposto originalmente por Coetzee (1998). Tal método, denominado *caixa limitante adaptativa*, emprega uma estrutura de busca no formato de um "L invertido".

O funcionamento do algoritmo de busca é dividido em três etapas:

- A estrutura no formato de "L invertido", que possui um tamanho inicial definido, é deslocada pela imagem de entrada, da esquerda para a direita, de cima para baixo. A cada nova posição, as seguintes condições são verificadas:
    - Todos os *pixels* sobre os quais a estrutura de busca está atualmente posicionada constituem o fundo da imagem?
    - Existe um ou mais *pixels* pertencentes a objetos, tanto na adjacência direita, quanto na adjacência inferior das barras vertical e horizontal, respectivamente, as quais compõem a estrutura de busca?

  Caso a resposta para ambas condições for verdadeira, a atual posição provavelmente consiste na borda esquerda superior de um caractere, conforme ilustrado na Figura 4.9(a);
- Uma segunda barra vertical é então deslocada para a direita, *pixel* a *pixel*, a partir da coluna que integra a estrutura de busca. Esta segunda barra vertical

segue sendo deslocada enquanto pelo menos um dos *pixels* que a constitui pertença a um objeto. Desta maneira é determinada a largura do possível caractere, conforme ilustrado na Figura 4.9(b);

- Finalmente, uma segunda barra horizontal, cuja largura é igual àquela apurada na etapa anterior, é deslocada para baixo, *pixel* a *pixel*, a partir da linha que integra a estrutura de busca. Esta segunda barra horizontal segue sendo deslocada enquanto pelo menos um dos *pixels* que a constitui pertença a um objeto. Claramente, desta maneira é determinada a altura do possível caractere, conforme ilustrado na Figura 4.9(c).

Após a determinação da posição e das dimensões do objeto, uma verificação é realizada no intuito de classificar o mesmo. Para que o objeto seja classificado como caractere, esse deve possuir um preenchimento igual ou superior a 15% da sua área, conforme ilustrado na Figura 4.9(d). Caso este critério não seja satisfeito, o objeto não é considerado como caractere e descartado.

(a) (b) (c) (d)

Figura 4.9 – Ilustração do funcionamento do algoritmo caixa limitante adaptativa.

4.2.1.4 Interpolação “Vizinho Mais Próximo”

O redimensionamento de uma imagem, tópico este abordado Seção 2.3.2.3, se dá por meio de uma transformação geométrica de escala na imagem original. Segundo Sonka (2007), uma transformação geométrica consiste numa função-vetor T que mapeia o *pixel (x,y)* para uma nova posição *(x',y')*, sendo a mesma definida pelas equações componentes

$$x' = T_x(x,y), \qquad y' = T_y(x,y) \tag{4.3}$$

e composta por duas etapas:

- Transformação de coordenada de *pixel*, que mapeia as coordenadas dos *pixels* da imagem de entrada para os pontos na imagem de saída;

- Localização do ponto na matriz digital de entrada que corresponde ao ponto transformado para a determinação do nível de brilho do mesmo, o que geralmente é realizado por meio da técnica de interpolação.

No que diz respeito à etapa de transformação de coordenada de *pixel*, no caso de uma transformação geométrica de escala, a função-vetor *T* é dada por

$$x' = ax$$
$$y' = by \quad \textbf{(4.4)}$$

Como resultado da primeira etapa tem-se um conjunto de pontos transformados, definidos por coordenadas *(x',y')* não-inteiras. O maior problema, neste caso, reside no fato da apuração do valor de um dado *pixel (x'',y'')* (pertencente à matriz discreta de saída) não ser possível pelo simples arredondamento das coordenadas não-inteiras *(x',y')*. Tal procedimento poderia causar tanto a atribuição de mais de um nível de brilho a um mesmo *pixel*, quanto a não-atribuição de valor a alguns deles (OWENS, 2007).

A solução consiste na apuração dos níveis de brilho a partir das coordenadas inteiras definidas na própria imagem de saída. Assim sendo, faz-se necessária a definição das coordenadas *(x,y)* correspondentes a cada um dos pontos na imagem de saída. Tais coordenadas de origem são computadas a partir da inversão da transformação dada pela Equação (4.3), resultando em

$$(x,y) = T^{-1}(x'',y'') \quad \textbf{(4.5)}$$

Em geral, as coordenadas *(x,y)* apuradas pela transformação inversa também não constituem pontos discretos na matriz de origem. Entretanto, dadas as matrizes discretas de origem $f_o(x,y)$ e de destino $f_d(x'',y'')$, os níveis de brilho $f_d(x'',y'')$ podem ser obtidos a partir da técnica de interpolação “vizinho mais próximo”. Para tanto, essa técnica atribui ao ponto *(x'',y'')* o nível de brilho do ponto discreto mais próximo a *(x,y)*. Desta forma, a interpolação “vizinho mais próximo” é dada por

$$f_d(x'',y'') = f_d(arredonda(x),arredonda(y)) \quad \textbf{(4.6)}$$

4.2.2 Reconhecimento dos caracteres

A segunda e última etapa do sistema aqui proposto consiste no reconhecimento dos objetos apurados na etapa anterior. Conforme ilustrado na Figura 4.4, tais objetos já se encontram descritos num formato próprio para utilização como entrada do processo de reconhecimento, ou seja, normalizados numa matriz de tamanho padrão.

De acordo com o disposto na Seção 2.5.7, os principais métodos empregados na tarefa de reconhecimento são o Estatístico (máquinas de suporte vetorial, análise de agrupamentos), o Conexionista (RNAs) e o Sintático (análise baseada em gramática).

Neste trabalho será empregado o método Conexionista na execução da tarefa de reconhecimento de caracteres. A escolha de RNAs para a execução desta tarefa pode ser justificada por uma série de razões, dentre as quais:

- RNAs têm sido utilizadas com sucesso na solução dos mais diversos problemas científicos e industriais relacionados à área da visão computacional (MOLZ, 2001) (JÄHNE, 2000);
- RNAs são capazes de solucionar problemas que por meio de métodos tradicionais são de difícil modelagem (MOLZ, 2001). Neste caso a solução é construída de forma iterativa a partir do treinamento da rede;
- A capacidade de generalização das RNAs permite que os padrões sejam corretamente identificados mesmo na presença de um certo nível de ruído. Entenda-se por ruído, neste caso, a má formação ou certa distorção no ângulo dos caracteres, por exemplo;
- Vários SRPLVs têm empregado RNAs como método de reconhecimento de caracteres, dentre os quais aqueles citados por Anagnostopoulos (2006), Bremananth (2005), Gesualdi (2002), Campos (2001), Yap (1999), Coetzee (1998), Draghici (1997) e Nijhuis (1995).

Conforme mencionado por Dias (2005), um dos problemas existentes em relação ao processo de reconhecimento de caracteres alfanuméricos diz respeito à semelhança na tipografia de caracteres diferentes. Como exemplo, pode-se citar o caso dos caracteres {‘0’, ‘O’, ‘Q’, ‘D’}, {‘2’, ‘7’, ‘Z’}, {‘4’, ‘A’}, {‘5’, ‘S’}, etc. No intuito de resolver os conflitos entre letras e números semelhantes, esta proposta emprega dois artifícios:

- A sintaxe da placa de licenciamento veicular brasileira (três letras seguidas por quatro números);
- RNAs distintas para letras e números, onde a seleção da rede utilizada é feita a partir da regra definida no item anterior (sintaxe da placa).

A partir da definição da existência de redes distintas para letras e números, já é possível esboçar a topologia para cada uma delas. Como neste caso optou-se pela utilização da codificação 1-de-C, onde cada classe é representada por um neurônio distinto, então a rede das letras terá 26 neurônios na camada de saída, enquanto a de números, 10. A camada de

entrada será idêntica para ambas as redes, possuindo, num primeiro momento, 225 neurônios (matriz de 15x15), conforme sugerido por (DRAGHICI, 1997) (COETZEE, 1998) (NI, 2007). Já o número de camadas escondidas, bem como o número de neurônios em cada uma delas será determinado a partir de simulações.

Em relação às etapas de treinamento e propagação, apenas a segunda necessita fazer parte do sistema em si. A justificativa para tal reside no fato de que uma vez definidos os pesos para cada sinapse, eles geralmente não são mais alterados, haja vista a capacidade de generalização de uma rede adequadamente treinada. Desta maneira, a etapa de treinamento será implementada como uma opção do *software* sobre plataforma x86, mas não constituirá o sistema de extração e reconhecimento de caracteres ópticos.

Dentre os diferentes tipos de RNAs, a do tipo PMC foi a selecionada para constituir o classificador alfanumérico do sistema. O aprendizado nas redes PMC é efetuado de modo supervisionado a partir do algoritmo de treinamento denominado retropropagação de erro. No que diz respeito à função de ativação dos neurônios, será utilizada a tangente hiperbólica, haja vista que funções anti-simétricas geralmente implicam num tempo de treinamento (número de iterações) menor do que as não simétricas, como a função logística (HAYKIN, 2002). A seguir é discutida a estrutura, bem como, principalmente, o algoritmo de treinamento de uma rede PMC.

4.2.2.1 RNAs *Perceptron* de Múltiplas Camadas

Conforme já mencionado na Seção 2.4.4, redes PMC são derivadas das redes *feedforward* de múltiplas camadas. Entretanto, por conta do aprendizado se dar por meio do algoritmo de retropropagação de erro, o fluxo de sinais nas redes PMC se dá em ambas as direções, conforme ilustrado na Figura 4.10. Desta maneira, os tipos de sinais existentes em tais redes são os seguintes:

- Sinais de Função: um sinal de função consiste num estímulo que, a partir da camada de entrada, é propagado para adiante, neurônio por neurônio, até a última camada, onde é tido como um sinal de saída. Um sinal de função assim se denomina pois (a) presume-se que realize alguma função significativa e (b) porque em cada neurônio que passa, o mesmo é calculado em função das entradas e pesos relativos àquele neurônio.
- Sinais de Erro: um sinal de erro se origina num neurônio da última camada e é propagado no sentido inverso ao do sinal de função, camada por camada, até a

primeira camada escondida. Um sinal de erro é assim denominado porque o cálculo do mesmo envolve uma função de erro dependente, de uma forma ou de outra.

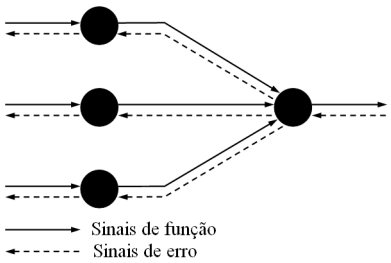


Figura 4.10 – Tipos de sinais numa rede PMC.

A existência de tais tipos de sinais em uma rede PMC se deve ao fato do treinamento da mesma ser realizado por intermédio do algoritmo de retropropagação de erro. Basicamente, tal algoritmo consiste em duas etapas. Na primeira delas são calculados os sinais de função dos neurônios da rede, processo este que ocorre na direção da entrada para a saída. Já na segunda etapa, a qual ocorre no sentido da saída para a entrada (daí o nome do algoritmo), são computados os sinais de erro. Tais sinais de erro são dados em função da diferença entre a saída atual e esperada de cada neurônio, sendo tais sinais utilizados no ajuste dos pesos das conexões, no intuito de reduzir o erro apurado.

De um modo formal, o aprendizado por retropropagação de erro pode ser explicado a partir do grafo ilustrado na Figura 4.11, onde o índice *i* diz respeito à camada de entrada, o índice *j* à camada escondida e o índice *k*, à camada de saída.

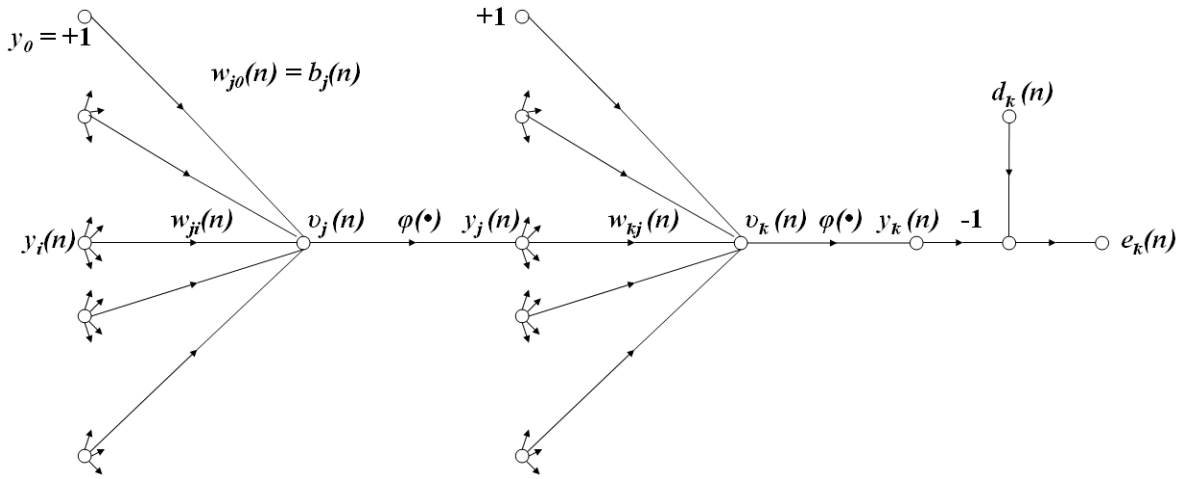


Figura 4.11 – Grafo de fluxo de sinal ilustrando os detalhes de uma conexão entre os neurônios *j* e *k* (HAYKIN, 1999).

Na primeira etapa do processo de aprendizado por retropropagação de erro, a computação do sinal de função de um dado neurônio *j* inicia pela apuração do campo local induzido do mesmo

$$\upsilon_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \qquad \textbf{(4.7)}$$

onde *m* é o número total de entradas (excluindo o *bias*) conectadas ao neurônio *j*. Desta forma, o valor do sinal de função deste neurônio para a iteração *n* (apresentação do *n*-ésimo padrão de treinamento) é dado por

$$y_j(n) = \varphi_j\left(\upsilon_j(n)\right) \qquad \textbf{(4.8)}$$

O mesmo processo de propagação em direção à camada de saída é realizada para o neurônio *k*, entretanto utilizando agora a saída $y_j(n)$ como entrada.

Após a propagação dos sinais de função por todos os neurônios da rede, ocorre a apuração do sinal de erro, a partir dos nodos de saída da mesma. Assim sendo, o sinal de erro para um neurônio de saída *k* é dado por

$$e_k(n) = d_k(n) - y_k(n) \qquad \textbf{(4.9)}$$

onde $d_k(n)$ é o valor desejado para o neurônio *k*. A partir de $e_k(n)$ torna-se possível a apuração do valor instantâneo da energia de erro para o neurônio *k*, dado por $1/2\, e_k^2(n)$. Em conseqüência, obtém-se o valor instantâneo da energia de erro total, o qual consiste no somatório dos valores instantâneos da energia de erro de todos os neurônios da camada de saída, expresso por

$$\xi(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n) \qquad \textbf{(4.10)}$$

onde *C* corresponde ao conjunto de neurônios na camada de saída. Finalmente, a energia de erro quadrática média, obtida pelo somatório dos $\xi(n)$ de todas as iterações *n* e normalizada em relação a *N* é dada por

$$\xi_{av} = \frac{1}{N} \sum_{n=1}^{N} \xi(n) \qquad \textbf{(4.11)}$$

onde *N* é o tamanho do conjunto de amostras utilizado no processo de aprendizado. Para um dado conjunto de treinamento, a energia de erro médio ξ_{av} representa a função de custo como uma medida de desempenho de aprendizado.

Durante o processo de aprendizado busca-se reduzir o valor de tal função de custo pelo ajuste gradual dos parâmetros livres da rede (pesos de sinapse e níveis de *bias*). Tais ajustes podem ser realizados após a apresentação de cada padrão de treinamento *n* (treinamento seqüencial), bem como somente após uma época, ou seja, após a apresentação de todo o

conjunto de treinamento (treinamento em lote). Independente do momento do ajuste, o mesmo ocorre no sentido inverso do utilizado na apuração do sinal de função e consiste na adição de uma parcela de ajuste ao peso de cada sinapse, parcela essa dada por

$$\Delta w_{kj}(n) = \eta \delta_k(n) y_j(n) \quad \textbf{(4.12)}$$

onde η consiste na taxa de aprendizado e $\delta_k(n)$ no gradiente local. No caso de um nodo de saída, o gradiente local é dado por

$$\delta_k(n) = e_k(n) \varphi'_k(\upsilon_k(n)) \quad \textbf{(4.13)}$$

onde $\varphi'_k(\upsilon_k(n))$ consiste na primeira derivada da função de ativação utilizada pelo neurônio. Já no caso de um neurônio *j*, situado numa camada escondida, o cálculo do gradiente local do mesmo não pode ser efetuado de forma direta, pois não há um valor de saída desejado para o mesmo, como ocorre no caso de um nodo de saída. Neste caso, o sinal de erro deste neurônio *j* deve ser apurado de forma recursiva em termos dos sinais de erro de todos os neurônios aos quais este neurônio escondido está diretamente conectado. Desta maneira, o gradiente local de um neurônio escondido *j* é dado por

$$\delta_j(n) = \varphi'_j(\upsilon_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad \textbf{(4.14)}$$

onde o sinal de erro do neurônio escondido *j* constitui-se no somatório dos produtos entre os gradientes locais e pesos das conexões de todos os neurônios conectados a *j* na direção da saída.

No que diz respeito à taxa de aprendizado, quanto menor o valor deste parâmetro, mais suave é a trajetória descrita no espaço de pesos durante o aprendizado e, conseqüentemente, maior o tempo de convergência do processo. Por outro lado, o aumento do valor deste parâmetro se traduz numa convergência mais rápida, ao passo que pode inserir instabilidade no processo. Uma maneira de acelerar o processo e ao mesmo tempo minimizar o risco de instabilidades consiste na inserção do *termo de momento* à parcela de ajuste de erro

$$\Delta w_{kj}(n) = \alpha \Delta w_{kj}(n-1) + \eta \delta_k(n) y_j(n) \quad \textbf{(4.15)}$$

onde α consiste num valor situado no intervalo]0,1], denominado *constante de momento*. A inclusão do termo de momento tende a acelerar a minimização da função de custo quando a parcela de ajuste possuir o mesmo sinal por diversas iterações, bem como a estabilizar possíveis oscilações causadas por consecutivas parcelas de ajuste com sinais contrários.

Como, em geral, a convergência do algoritmo de retropropagação de erro não pode ser demonstrada, não há um critério bem definido para determinar a interrupção do mesmo. Neste sentido, como o processo de atualização dos pesos tem de ser interrompido em algum

momento, existem alguns critérios lógicos que podem ser empregados na determinação do momento da interrupção, dentre os quais:

- Norma Euclidiana: segundo Kramer e Sangiovanni-Vincentelli (1989 apud HAYKIN, 1999), considera-se que o algoritmo de retropropagação de erro convergiu quando a norma Euclidiana do vetor de gradiente ($\partial\xi(n)/\partial w_{kj}(n)$) alcança um limite de gradiente suficientemente pequeno. O principal empecilho deste critério consiste na possibilidade de um tempo longo de aprendizado.
- Erro quadrático médio (EQM): considera-se que o algoritmo de retropropagação de erro convergiu quando a taxa absoluta de alterações no erro quadrático médio (ξ_{av}) por época é suficientemente pequena. Por *suficientemente pequena* entende-se uma taxa de variação entre 0,1% e 1% por época, embora taxas tão pequenas quanto 0,01% possam ser utilizadas. O empecilho deste critério consiste na possibilidade de término prematuro do processo de aprendizado.
- Parada prematura: outro critério empregado consiste na verificação do desempenho de generalização alcançado no processo de aprendizado a cada iteração. Durante o treinamento de uma rede neural, o erro gerado pela mesma tende a iniciar em um valor alto, baixar rapidamente e então continuar diminuindo de uma forma mais lenta conforme as iterações ocorrem. Entretanto, a partir de um certo número de iterações a rede começa a se tornar viciada (*overfitted*/*overtrained*) em relação ao conjunto de treinamento, o que diminui a capacidade de generalização da mesma. Uma solução simples, porém eficaz, consiste no emprego do método de parada prematura. Inicialmente, o conjunto de treinamento é dividido em dois subconjuntos distintos: estimativa e validação. A rede é treinada normalmente a partir do subconjunto de estimativa, sendo o erro apurado para ambos os subconjuntos. Conforme as iterações vão se sucedendo, os erros apurados para ambos os subconjuntos são minimizados, sendo a curva referente ao subconjunto de estimativa a mais acentuada. No momento em que a rede começa a se tornar viciada, o erro referente ao subconjunto de estimativa continua a diminuir, enquanto que o apurado para o subconjunto de validação tende a aumentar, conforme ilustrado na Figura 4.12. Quando esta condição é detectada, o treinamento é

interrompido e os pesos referentes ao momento de erro mínimo para o subconjunto de validação são recuperados.

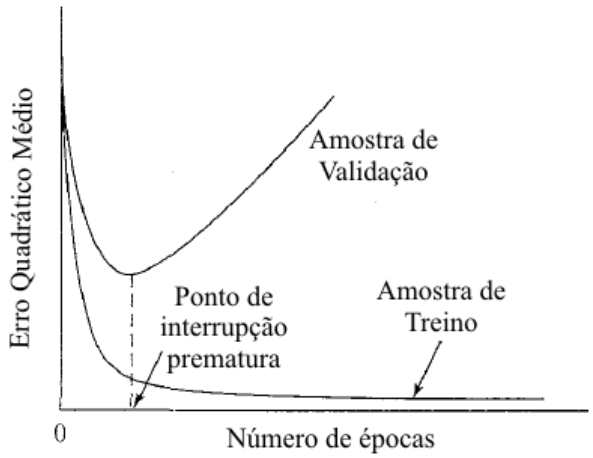


Figura 4.12 – Ilustração do procedimento de parada prematura do algoritmo de retropropagação de erros.

No próximo capítulo são discutidos os detalhes referentes à implementação do sistema descrito neste.

5 IMPLEMENTAÇÃO DO SISTEMA PROPOSTO

5.1 Introdução

A partir da teoria e da proposta discutidas nos capítulos anteriores, ocorreu a implementação dos sistemas. Inicialmente foi construída uma versão totalmente em *software* sobre plataforma x86, no intuito de avaliar, naquele momento, quais os métodos mais adequados para compor o sistema embarcado. A partir dos resultados obtidos pela implementação em *software* foi então projetada e implementada a versão embarcada do sistema.

Assim sendo, neste capítulo são discutidos os detalhes referentes às implementações, a partir da seqüência mencionada no parágrafo anterior. Resultados parciais são apresentados conforme as etapas se sucedem, no intuito de ilustrar as dificuldades e justificar as escolhas efetuadas. Os resultados finais, entretanto, são apresentados no próximo capítulo, limitando-se este na implementação em si.

5.2 Sistema sobre plataforma x86

5.2.1 Introdução

A versão de avaliação do sistema desenvolvido sobre plataforma x86 é composta por duas opções principais, além de algumas outras secundárias. Dentre as opções principais, a primeira implementa as tarefas relativas à extração de caracteres, enquanto a outra possibilita o reconhecimento dos mesmos, a partir dos resultados gerados pela primeira. Já em relação às opções secundárias, as mesmas foram criadas no intuito de possibilitar a execução de tarefas auxiliares, como as de configuração, geração de dados e apuração de resultados. Inicialmente, as duas tarefas principais, embora complementares, foram implementadas como opções distintas, no intuito de facilitar o processo de avaliação e a própria construção das mesmas. Neste sentido, efetuou-se a comunicação entre as duas tarefas por intermédio de arquivos de texto, até porque a etapa de localização de placas já fornecia saídas neste formato.

Na seqüência, cada uma das duas opções principais é discutida em detalhes, culminando a seção referente à versão do sistema sobre plataforma x86 na apresentação da configuração final do mesmo.

5.2.2 O módulo de extração dos caracteres

A opção de extração de caracteres foi construída de modo a permitir a avaliação dos diferentes métodos mencionados na Seção 4.2.1, bem como possibilitar a atribuição do respectivo código ASCII a cada um dos caracteres ópticos localizados. Apesar do processo de atribuição dos códigos ASCII estar relacionado à tarefa de reconhecimento de caracteres, inicialmente foi necessária a realização da mesma por meio de intervenção humana. A justificativa para tal reside no fato da necessidade de entradas consistentes para a posterior construção da versão automatizada deste processo.

Ambas as tarefas de avaliação e atribuição de códigos ASCII foram realizadas a partir da interface ilustrada na Figura 5.1, estando os parâmetros de entrada do processo situados na região "A". Dentre tais parâmetros, destacam-se os algoritmos de binarização e de identificação de objetos e a opção de ajuste dos objetos encontrados

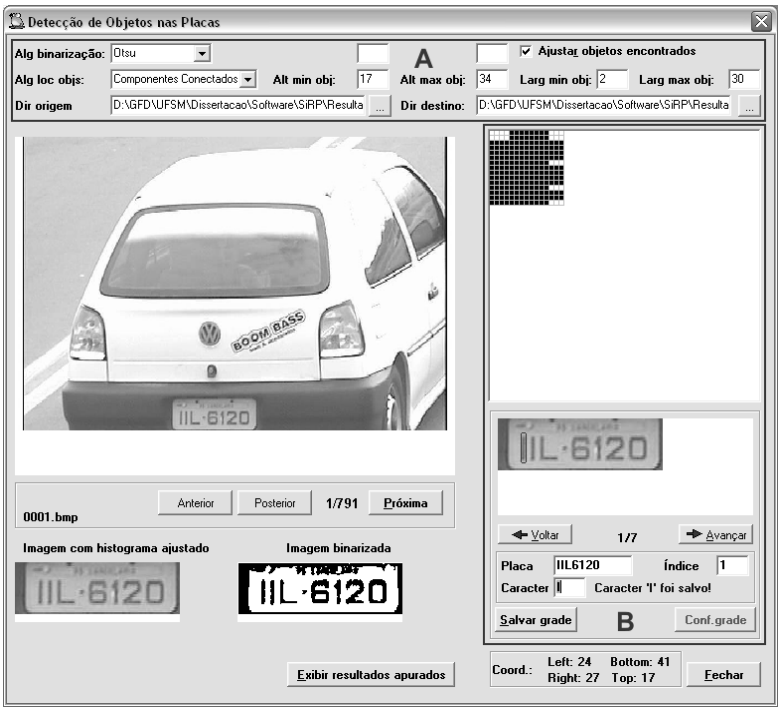


Figura 5.1 – Interface da opção que implementa o processo de extração de caracteres, na versão de avaliação do sistema construído sobre plataforma x86.

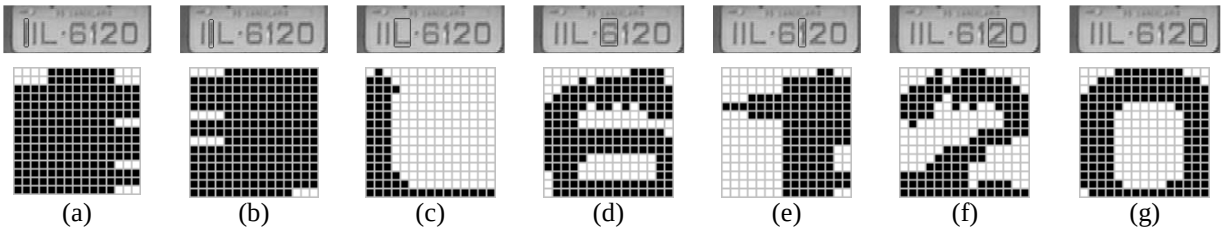
No que diz respeito aos algoritmos de binarização, foram avaliados ambos os métodos discutidos previamente na Seção 4.2: Otsu e Niblack. O mesmo não ocorreu em relação aos algoritmos de identificação de objetos, haja vista que dos três métodos previamente selecionados e discutidos na Seção 4.2.1, apenas dois foram de fato implementados, conforme

detalhado na Seção 5.2.2.5. Já a possibilidade de não realizar o ajuste dos objetos foi disponibilizada apenas no intuito de permitir a eventual visualização de todos os objetos localizados. Na prática, tal ajuste sempre deve ser feito, pois o mesmo envolve o processo de classificação dos objetos, imprescindível para o correto funcionamento do processo. Juntamente à classificação foram implementados os métodos de ordenação e localização forçada de objetos, os quais não haviam sido previstos anteriormente e são detalhados na Seção 5.2.2.3.

O processamento realizado por intermédio da opção inicia pela carga da primeira entrada contida no diretório de origem. Cada entrada consiste num par de arquivos, sendo tal par composto pela imagem do veículo em trânsito e as coordenadas da região que contém a placa do mesmo.A partir do conteúdo destes dois arquivos se obtém uma nova imagem em 256 tons de cinza, contendo apenas a região da placa, a qual é binarizada a com base no algoritmo atualmente selecionado. A imagem binarizada é então utilizada como entrada no processo de identificação de objetos, igualmente realizado pelo respectivo algoritmo atualmente selecionado. Como resultado, tem-se um conjunto de coordenadas referentes a todos os objetos encontrados. Cada elemento deste conjunto é classificado de acordo com os critérios dimensionais previamente especificados, permanecendo no conjunto apenas aqueles que satisfaçam tais critérios.

A partir deste conjunto de coordenadas de caracteres é que se dá a intervenção humana. Para tanto, uma cópia da imagem original da região da placa é utilizada como base na projeção dos retângulos que delimitam cada caractere encontrado, conforme ilustrado na região “B” da Figura 5.1 Conforme os objetos são selecionados, a respectiva saída normalizada é gerada e exibida logo acima da imagem da placa. Ao operador humano, cabe a tarefa de informar o código ASCII da placa identificada pelo mesmo no campo correspondente e certificar-se de que a seqüência dos objetos no conjunto e a localização dos retângulos na imagem da placa identificam corretamente o respectivo código ASCII. Tal processo é ilustrado na Figura 5.2, onde cada item é obtido a partir de uma interação do operador humano.

Ao término do processamento, têm-se dois arquivos de resultado para cada entrada. O primeiro consiste no arquivo original de coordenadas da região da placa, acrescido das coordenadas de cada objeto localizado em tal região. Já o segundo arquivo contém as versões normalizadas de cada objeto, as quais são utilizadas posteriormente no processo de treinamento da RNA.



(a) (b) (c) (d) (e) (f) (g)

Figura 5.2 – Ilustração do resultado do processo de localização e normalização de caracteres.

5.2.2.1 Arquivos de entrada

Conforme comentado na seção anterior, cada entrada é composta por um par de arquivos, sendo o primeiro deles a imagem do veículo em trânsito e o segundo um conjunto de coordenadas que definem uma ou mais regiões retangulares no primeiro arquivo.

No que diz respeito ao arquivo de imagem, as características do mesmo são as seguintes:

- Dimensões: 800x600 *pixels*;
- Formato: BMP;
- Modelo de cores: RGB (embora utilize apenas 256 tons de cinza) ;
- Intensidade de *bits*: 24 *bits*/*pixel* (embora oito *bits*/*pixel* seja suficiente).

Já em relação ao arquivo de coordenadas, o mesmo consiste num arquivo de texto comum, onde cada entrada é composta por um identificador numérico distinto, bem como as quatro coordenadas que definem a região retangular, conforme ilustrados na Figura 5.3.

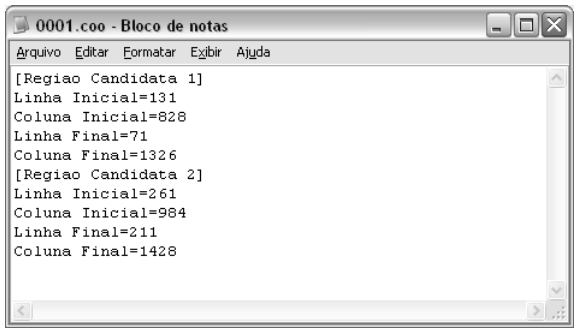


Figura 5.3 – Ilustração do conteúdo de um arquivo de coordenadas das regiões candidatas, fornecido pelo módulo desenvolvido por Pacheco (2007).

O arquivo de coordenadas acima se refere ao processamento ilustrado na Figura 5.1, onde a primeira região candidata diz respeito à placa processada naquela mesma figura. Diferentemente deste trabalho, a origem do sistema de coordenadas utilizado na geração do arquivo acima está situada no canto direito inferior, conforme indicado pelo valor das

coordenadas inicial e final do eixo *y*. Entretanto, tal questão é de trivial solução, bastando subtrair o valor das mesmas da altura da imagem para que se tenha o valor correto em relação à uma origem situada no canto esquerdo superior. Em relação ao eixo *x*, também é necessária uma adequação das coordenadas, haja vista que as mesmas levam em consideração *pixels* de três *bytes* (24 *bits*). Neste caso, uma simples divisão por três soluciona a questão.

5.2.2.2 Binarização

Conforme discutido na Seção 4.2, por conta da diferença de representação entre as imagens fornecidas e as aceitas como entradas do sistema, fez-se necessária a implementação de um processo inicial de conversão. Neste sentido, as técnicas de limiarização de Otsu e de Niblack foram previamente selecionadas e posteriormente avaliadas.

Inicialmente ocorreu a implementação de ambos os algoritmos. O algoritmo que implementa a limiarização global de Otsu foi adaptado a partir do código fonte disponibilizado pela versão 3.45 do *software* XITE (XITE, 2005). Já o algoritmo de limiarização local de Niblack foi implementado a partir da equação citada por Leedham (2003). Em seguida foi criada uma opção específica no *software* implementado sobre plataforma x86, ilustrada na Figura 5.4, no intuito de permitir a comparação simultânea dos resultados obtidos pela execução dos algoritmos. Dentre os critérios levados em consideração na avaliação podem ser citados a boa formação dos objetos, o número de objetos resultantes e a ausência de conexões indevidas entre os objetos.

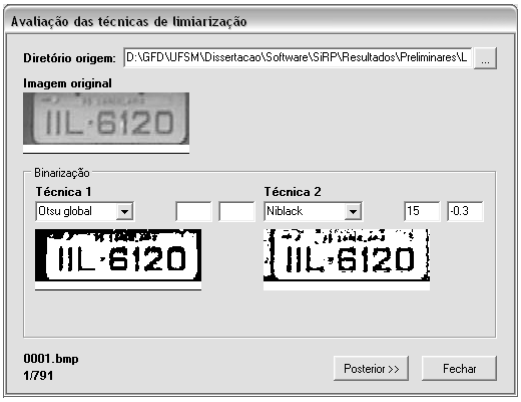


Figura 5.4 – Interface de avaliação das técnicas de limiarização.

Após o término da avaliação, a qual utilizou como base todas as 791 imagens disponíveis inicialmente, constatou-se que:

- na maioria dos casos o método de Otsu gera um número menor de objetos (Figura 5.5(a)-(b)), evitando assim a conexão indevida entre as bordas da placa e os caracteres (Figura 5.5(c));
- em alguns casos, como o ilustrado na Figura 5.5(d), o método de Niblack consegue desfazer conexões indevidas já presentes na imagem original;
- no restante dos casos os resultados obtidos por ambos os métodos se equivalem (Figura 5.5(e)).

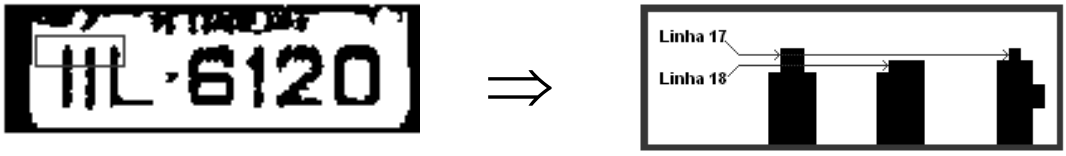
Com base nestas considerações, a técnica de Otsu foi escolhida, haja vista que na maioria dos casos gera um resultado superior ou igual ao obtido por Niblack, com a vantagem de ser mais simples, rápida e gerar um número inferior de regiões.

(a) (b) (c) (d) (e)

Figura 5.5 – Alguns dos resultados obtidos a partir das técnicas de limiarização avaliadas. As imagens originais (256 tons de cinza) estão dispostas na primeira linha, enquantos as obtidas pela técnica de Otsu na segunda e por Niblack na terceira.

5.2.2.3 Ordenação e localização por força bruta

Durante a implementação do processo de extração de caracteres, notou-se a necessidade da criação de uma etapa intermediária de ordenação dos objetos localizados. Tal etapa demonstrou-se necessária por conta da ordenação inicial dos objetos, gerada durante o processo de localização, que nem sempre respeitava a ordem na qual eles se encontravam dispostos na placa, comprometendo assim o reconhecimento da mesma. A causa desta possível ordenação inicial incorreta reside no fato ilustrado na Figura 5.6. Após a limiarização de uma placa, pode ocorrer da linha inicial de um determinado objeto ser inferior à de um objeto anterior, conforme ilustra a Figura 5.6(b), que consiste no detalhe ampliado da região contida no retângulo na Figura 5.6(a). Como os métodos etiquetação de componentes conectados e caixa limitante adaptativa processam a imagem da esquerda para a direita, de cima para baixo, então a ordem inicial do terceiro caractere é inferior à do segundo. A solução consiste na simples ordenação dos objetos, após o término do processo de localização dos mesmos, a partir da coordenada inicial no eixo *x* de cada um deles.



(a) (b)

Figura 5.6 – Ilustração do fato que originou a criação da etapa de ordenação dos objetos.

O outro método implementado durante a construção do módulo de extração de caracteres foi o de localização de caractere por força bruta, inspirado no citado por Souza (2000). Tal método pode ser utilizado no caso de um número inferior a sete caracteres serem localizados numa placa, conforme ilustrado na Figura 5.7(a), cujo motivo causador desta circunstância, por exemplo, é a conexão indevida de um objeto, conforme a Figura 5.7(b).

A implementação realizada neste trabalho difere da original, haja vista que a última se baseia na idéia de que o espaço que separa as letras dos números é maior do que aqueles que separam os caracteres em si, suposição esta que em muitas situações se demonstrou falsa. Assim sendo, o método aqui implementado inicialmente apura a altura e a largura média dos caracteres encontrados, para depois localizar a região de maior distância entre dois caracteres e então delimitá-la pela dimensão média apurada anteriormente. No caso da Figura 5.7(a), tal região se situa entre os caracteres “6” e “9”, onde está localizado o caractere “3”.

(a) (b)

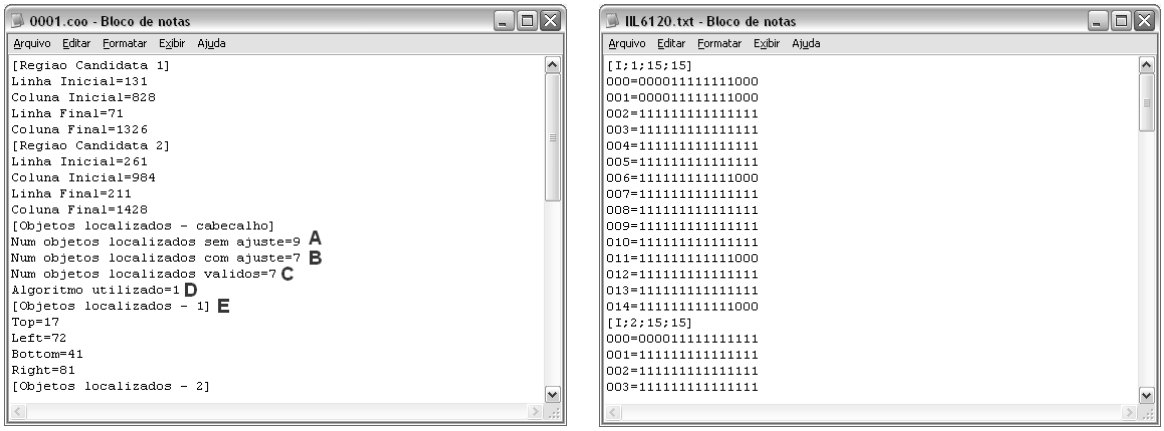
Figura 5.7 – Exemplo de emprego do método de localização de caracteres por força bruta.

5.2.2.4 Arquivos de saída

Como resultado do processo de extração dos caracteres de uma imagem, discutido na Seção 5.2.2, têm-se o par de arquivos ilustrados na Figura 5.8. O arquivo exibido na Figura 5.8(a) consiste no arquivo de entrada do processo (Figura 5.3), acrescido de informações referentes aos objetos localizados. Estas informações dizem respeito ao número de objetos encontrados (item “A”), de caracteres encontrados (item “B”), de caracteres válidos (item “C”), bem como o algoritmo de localização de objetos utilizado (item “D”). Além disso, também são armazenadas as coordenadas de cada caractere válido encontrado (item “E”).

A questão da validade do caractere está relacionada ao algoritmo de localização de caracteres por força bruta. No intuito de avaliar o desempenho do mesmo, um segundo julgamento foi realizado por meio de intervenção humana a partir dos objetos classificados como caracteres pelo sistema. O número de objetos que eram de fato caracteres, proveniente do segundo julgamento, foi então armazenado junto ao arquivo de resultados para posterior análise.

Já no que diz respeito ao arquivo ilustrado na Figura 5.8(b), o mesmo é constituído pelas versões normalizadas dos caracteres localizados. Cada entrada do arquivo é composta por um cabeçalho e um respectivo conjunto de dados. No cabeçalho estão armazenados o código ASCII do caractere, a posição que ele ocupa na placa, bem a altura e a largura da matriz utilizada na normalização do mesmo. Na seqüência encontra-se armazenado o conteúdo da matriz de normalização. Tal arquivo serve como entrada no processo de geração de dados para o treinamento da RNA, o qual é discutido na Seção 5.2.3. Para constar, a primeira entrada do arquivo ilustrado na Figura 5.8(b) se refere à Figura 5.2(a).



(a) (b)

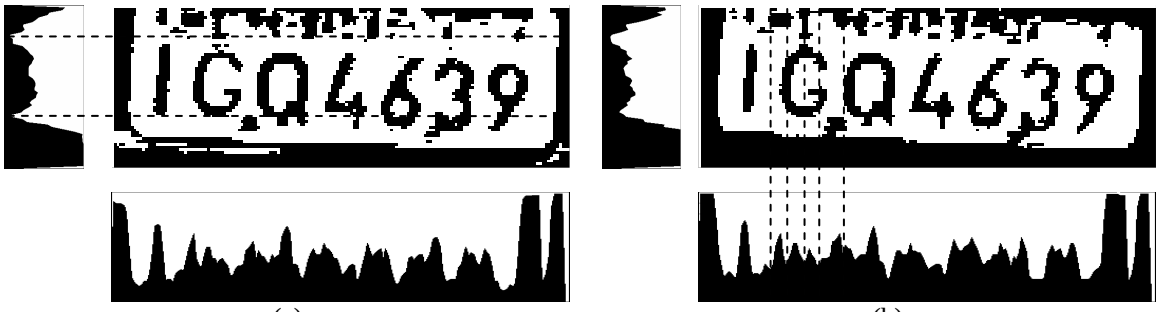
Figura 5.8 – Arquivos resultantes do processo de extração de caracteres.

5.2.2.5 Identificação de regiões

A implementação e posterior avaliação dos métodos de identificação de regiões constituiu a principal tarefa executada durante o processo de construção da opção de extração de caracteres sobre a plataforma x86. Como o intuito era o de construir a versão em *hardware* do método selecionado, considerável atenção foi dedicada à escolha do mesmo, haja vista que o método a ser empregado na construção do outro principal bloco de *hardware* (normalização) já estava definido.

O processo de seleção iniciou-se então pela implementação dos métodos de etiquetação de componentes conectados e caixa limitante adaptativa, a partir das propostas de Yang (2005) e Souza (2000), respectivamente. Já o método das projeções horizontal e vertical, apesar de inicialmente considerado, teve sua implementação cancelada após testes iniciais de viabilidade.

O motivo do descarte do método das projeções diz respeito à existência de ruído nas extremidades das regiões utilizadas como entrada do sistema, situação esta distinta da considerada inicialmente (Figura 4.8). Este ruído surge durante o processo de limiarização, quando um número considerável de *pixels*, cujo nível de cinza é igual ou próximo ao daqueles que formam os caracteres, existe nas extremidades da região retangular. Desta maneira, tais *pixels* marginais acabam por compor objetos que formam uma espécie de "moldura" ao redor dos caracteres, interferindo no resultado das projeções, conforme ilustrado na Figura 5.9. Na Figura 5.9(a) nota-se o erro de enquadramento vertical introduzido pelas áreas de ruído lateral, enquanto na Figura 5.9(b) é visível a dificuldade de determinação da largura dos caracteres, como exemplificado pela linhas horizontais tracejadas.



(a) (b)

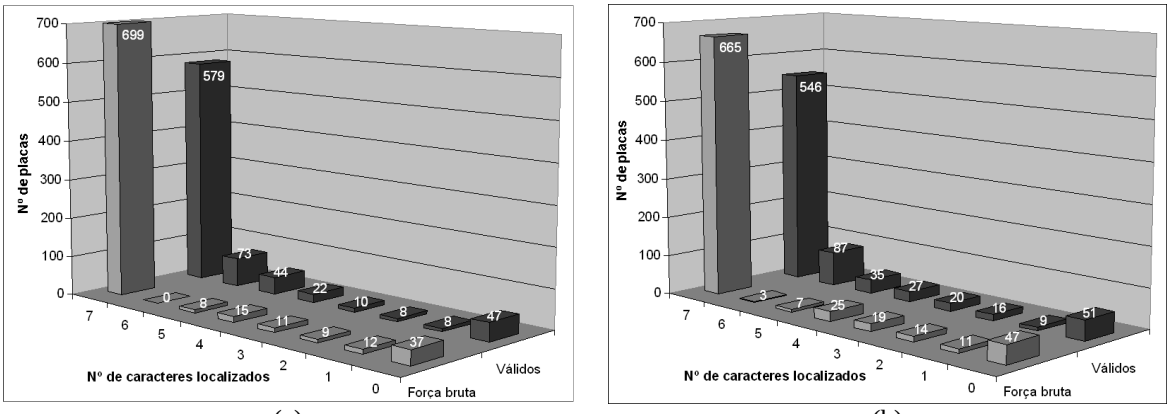
Figura 5.9 – Ilustração das dificuldades encontradas durante avaliação do método de projeções horizontal e vertical.

No que se refere à relação entre a técnica de limiarização utilizada e a existência do ruído de limiarização, a mesma demonstrou-se de média significância neste caso. Ao mesmo tempo que o método de Otsu costuma gerar menos objetos (vide Figura 5.5), existem inúmeras situações onde o nível de ruído gerado por ambos os métodos é similar. Uma desta situações é o caso dos resultados ilustrados na Figura 5.9(a) e na Figura 5.9(b), gerados por Niblack e Otsu, respectivamente. Dentre as soluções que poderiam ser adotadas no intuito de reduzir o nível de ruído e possibilitar o uso do método das projeções, uma delas consiste no uso da etapa de enquadramento, discutida na Seção 2.5.7. Entretanto, tal solução além de complexa, situa-se além do escopo deste trabalho.

Seguindo com o processo de seleção do método de localização de caracteres, após a implementação dos dois que ainda se mantinham candidatos, cada um deles foi utilizado no processamento do conjunto de imagens inicial. O fluxo de execução se deu conforme descrito nas seções anteriores, imagem a imagem, havendo intervenção humana no julgamento do número efetivo de caracteres localizados pelo sistema e na atribuição dos respectivos códigos ASCII aos caracteres válidos.

Os resultados referentes ao desempenho de cada um dos métodos foram a apurados a partir da opção existente na interface ilustrada na Figura 5.1 (“Exibir resultados apurados”). Tal opção processa os arquivos de extensão “coo” gerados como saída do processo de localização, extraindo de cada um deles o número de objetos localizados (considerando o uso do método de localização por força bruta) e o número de caracteres válidos (itens “B” e “C” na Figura 5.8(a), respectivamente). Estes dados são então sintetizados, dando origem a duas listas distintas (caracteres encontrados e válidos), cujo conteúdo consiste na contagem de placas em função do número de caracteres encontrados.

Com base nestas listas foram gerados os gráficos de desempenho ilustrados na Figura 5.10(a) e na Figura 5.10(b), os quais se referem aos resultados apurados para o método de etiquetação de componentes conectados e para o método caixa limitante adaptativa, respectivamente. Os resultados obtidos são praticamente os mesmos para ambos os casos, existindo uma pequena vantagem no caso do método de etiquetação de componentes conectados. Um ponto negativo, também em ambos os casos, diz respeito ao número de caracteres inválidos encontrados, cuja principal causa diz respeito ao desempenho da implementação do método de localização por força bruta.



(a) (b)

Figura 5.10 – Resultados parciais obtidos a partir dos métodos (a) etiquetação de componentes conectados e (b) caixa limitante adaptativa.

5.2.2.6 Versão final

A partir dos resultados discutidos nas seções anteriores, a versão final do módulo de extração de caracteres sobre plataforma x86 foi definida conforme ilustração na Figura 5.11. No tocante ao método de localização de caracteres, apesar da necessidade de implementação de um processo adicional para a extração das coordenadas dos objetos localizados pelo mesmo, a etiquetação de componentes conectados foi o escolhido. Dentre os critérios utilizados na seleção do método pode-se citar os seguintes:

- Desempenho ligeiramente superior do método selecionado em relação ao outro avaliado;
- O fato do método caixa limitante adaptativa por si só não ser capaz de determinar os *pixels* que constituem o caractere no interior do retângulo que o delimita. Considerando o caso de existência de pequenos objetos nos limites de tal retângulo, é necessária a utilização de um método auxiliar, conforme indicado por Souza (2000), no intuito de recuperar apenas os *pixels* que compõem o objeto. Já no caso da etiquetação de componentes conectados essa possibilidade de distinção é inerente ao método, haja vista que cada objeto é identificado por um número distinto;
- O fato da existência de uma arquitetura de *hardware* razoavelmente documentada que implementa o método de etiquetação de componentes conectados de forma otimizada (YANG, 2005).

Em relação à tentativa de utilização do método de localização de caracteres por força bruta, preferiu-se abrir mão do mesmo em vista do elevado número de casos de falso positivo encontrados, onde uma possibilidade é dada como verdadeira, quando na verdade não o é. Provavelmente o desempenho insatisfatório está relacionado com a implementação aqui realizada do método, onde se tentou evitar um número elevado de consistências, no intuito de simplificar uma futura implementação em *hardware*.

Como a última das modificações realizadas à configuração inicial do módulo, têm-se a inclusão do módulo de ordenação dos caracteres encontrados, conforme discutido anteriormente, na Seção 5.2.2.3.

Na seqüência é discutida a implementação realizada no intuito de possibilitar o reconhecimento dos caracteres localizados pelo módulo recém descrito.

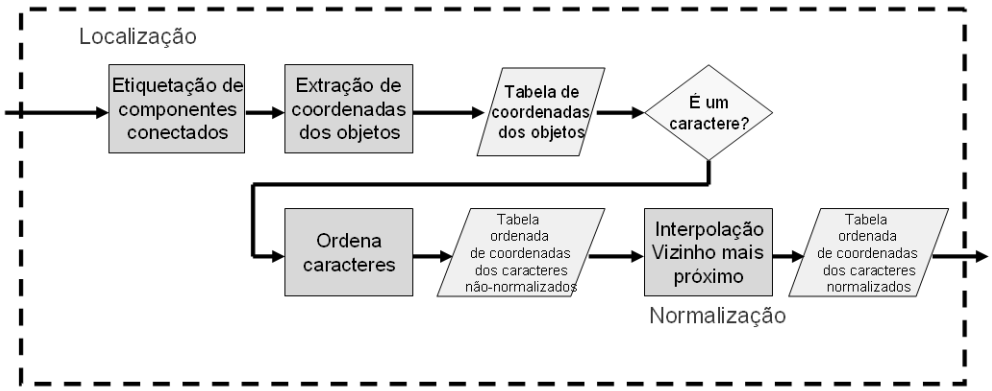


Figura 5.11 – Versão final do módulo de extração de caracteres sobre plataforma x86.

5.2.3 O módulo de reconhecimento de caracteres

De maneira similar à realizada na etapa de extração de caracteres, o módulo de reconhecimento foi construído no intuito de permitir a determinação da topologia de rede que possibilita o melhor desempenho para cada um dos casos (letras e números). Por conta da inexistência de uma equação que defina a topologia ótima da rede para um dado problema, a determinação da mesma é feita por intermédio de heurísticas e sucessivas tentativas a partir de diferentes configurações (número de neurônios escondidos). Dentre as heurísticas utilizadas podem ser citadas a experiência prévia em problemas similares e o número de classes que a rede deve ser capaz de distinguir. Geralmente quanto maior o número de classes que a rede distingue, maior é o número de neurônios escondidos necessários para possibilitar a convergência da mesma. No que diz respeito à avaliação do desempenho da topologia, a mesma pode ser realizada a partir do percentual de reconhecimento alcançado pela rede, haja vista que este é o objetivo de uma RNA utilizada como um classificador de padrões (HAYKIN, 1999).

Com base nestas considerações, a opção de reconhecimento foi implementada de modo a permitir o treinamento e a avaliação de desempenho de um RNA a partir de uma topologia selecionada. Inicialmente a interface ilustrada na Figura 5.12 é utilizada na realização do treinamento da rede, onde o algoritmo de aprendizado utilizado é o de retropropagação de erro, discutido na Seção 4.2.2.1. Tal algoritmo foi implementado a partir do código fonte utilizado por Molz (2001), o qual emprega o EQM como critério de parada, não utiliza o termo de momento e possibilita o ajuste da taxa de aprendizado de forma manual. A função de ativação utilizada foi a tangente hiperbólica, enquanto os pesos das sinapses utilizam representação numérica de ponto flutuante.

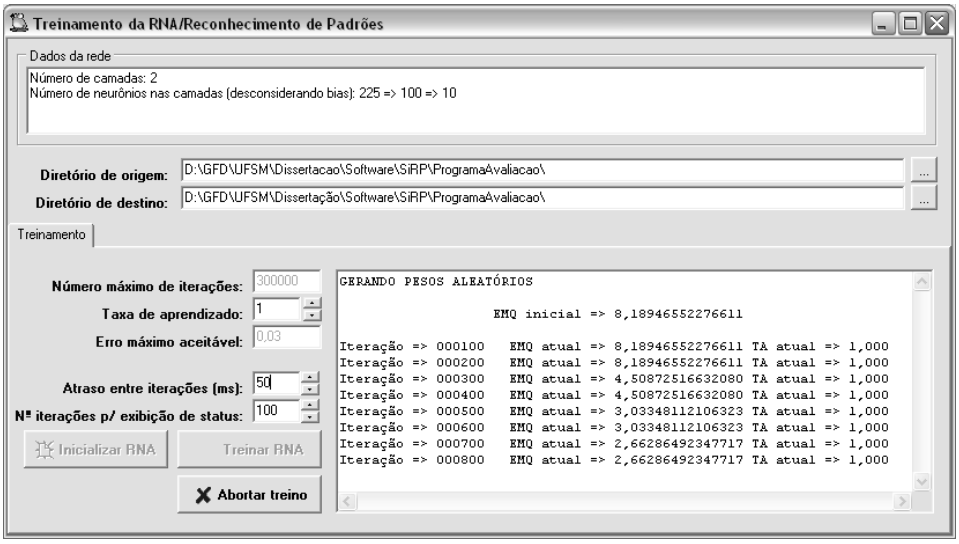


Figura 5.12 – Interface de treinamento da RNA implementada sobre plataforma x86.

Para a realização do treinamento, primeiramente faz-se necessário o ajuste dos parâmetros do processo. Dentre os parâmetros mais relevantes estão o número máximo de iterações, utilizado em conjunto com o EQM como critério de parada, a taxa de aprendizado e o valor do próprio EQM. Após o ajuste dos parâmetros, o próximo passo consiste na atribuição de pesos iniciais aleatórios para cada sinapse existente na rede (botão “Inicializar RNA” na Figura 5.12). Estando os parâmetros ajustados e os pesos iniciais atribuídos às sinapses, o processo de treinamento é então iniciado. Neste momento, a primeira ação executada é a leitura do arquivo de padrões de treinamento, cujo conteúdo é discutido na Seção 5.2.3.2.

Cada iteração do treinamento corresponde à apresentação de um padrão à rede, seguida pelo ajuste dos pesos das conexões em função do erro encontrado na camada de saída. Tal modo de treinamento é denominado *seqüencial*, em contraste ao modo *em lote*, onde o ajuste dos pesos é efetuado apenas ao final de uma época (o conjunto de todos os padrões utilizados no treinamento). Durante a execução do processo, informações a respeito do EQM e da taxa de aprendizado atuais são exibidas na tela, no intuito de permitir o ajuste manual da última.

O ajuste manual da taxa de aprendizado é realizado com o objetivo de acelerar o processo de convergência da rede, tendo como critério para o ajuste o valor do EQM. Caso tal valor permaneça constante durante sucessivas iterações, significa que o valor da taxa de aprendizado deve ser aumentado, possibilitando assim a variação do EQM. Já quando o valor do EQM varia durante as iterações, o valor da taxa de aprendizado deve ser mantido baixo

(geralmente menor que 1), no intuito de manter a curva do EQM na direção descendente. O inconveniente da realização do processo de forma manual é a constante atenção dispendida, bem como a possibilidade de conduzir a rede a uma situação em que não convirja nunca.

As iterações de treinamento se sucedem até o momento em que um dos critérios de parada é alcançado: número de iterações ou EQM. É desejado que o treinamento encerre a partir do segundo critério, haja vista que um valor de EQM superior ao especificado após o número máximo de iterações ter sido alcançado geralmente indica que o desempenho da rede é baixo. Levando em consideração o término do treinamento pela obtenção de um EQM inferior ao especificado, um arquivo contendo os atuais pesos das sinapses é criado e a etapa de avaliação do desempenho da rede é habilitada.

A etapa de avaliação de desempenho foi implementada junto à opção utilizada inicialmente para o treinamento da rede, conforme ilustrado na Figura 5.13, haja vista que os processos são de natureza complementar. Como primeiro passo na execução de uma avaliação ocorre a carga de um arquivo de padrões para a grade existente na interface. Tal arquivo é similar ao utilizado no treinamento no que diz respeito ao formato e número de entradas, mas difere em relação ao conteúdo, pois não possui nenhuma das entradas utilizadas no processo anterior. A utilização de um arquivo contendo entradas não utilizadas anteriormente visa apurar a capacidade de generalização da rede recém treinada.

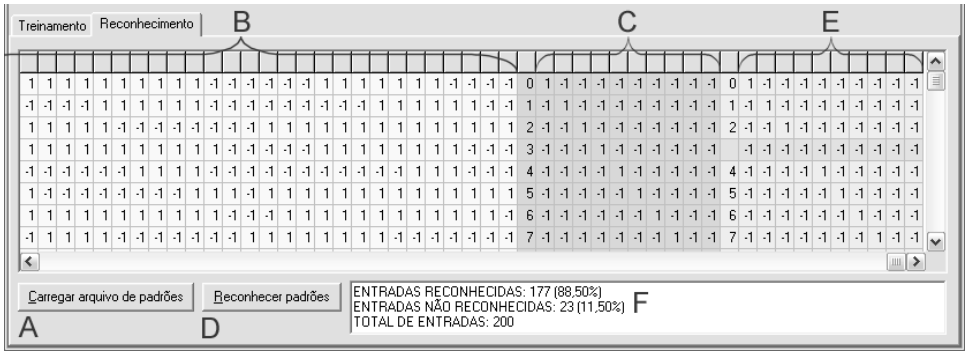


Figura 5.13 – Opção construída para avaliação da RNA.

Após a carga do arquivo de padrões para a grade, cada linha da mesma contém uma entrada a ser avaliada. Cada entrada, por sua vez, é composta pelo padrão de entrada, indicado parcialmente pelo item “B” Figura 5.13, bem como a respectiva saída esperada, indicada pelo item “C”. O processo de reconhecimento se dá pela propagação de cada padrão de entrada contido na grade pela RNA, seguido da comparação do resultado existente na camada de saída da rede (item “E”) com o padrão de saída esperado. As situações onde o resultado esperado

não condiz com o obtido são destacadas na grade por intermédio de células de fundo vermelho, no intuito de possibilitar a identificação das mesmas. O desempenho da RNA é expresso nas formas ordinal e percentual, tanto para o casos positivos, quanto para os negativos.

Posteriormente à implementação da interface de treinamento e reconhecimento, tomou-se conhecimento da existência do suporte ao projeto de RNAs disponibilizado pelo ambiente MATLAB. Apesar do esforço já empregado na construção da opção sobre plataforma x86, optou-se pelo uso do ambiente MATLAB a partir de então na realização das tarefas de treinamento e avaliação. Desta forma, da implementação original foram mantidas as opções de configuração, geração do arquivo de padrões, formato do arquivo de pesos das sinapses, bem como a implementação da RNA em si.

Dentre os motivos que levaram à opção pelo ambiente MATLAB na definição das duas topologias ótimas, encontram-se o amplo suporte fornecido ao projeto de RNAs, bem como o fato do ambiente ser próprio para a realização de simulações. No que diz respeito ao amplo suporte, destaca-se a existência de variantes do algoritmo de retropropagação de erro (ajuste automático da taxa de aprendizado, termo de momento, etc.). Já em relação ao fato do ambiente ser próprio para simulação, destacam-se a agilidade na construção e o reduzido tempo de validação de um protótipo. Tais tarefas demandam um maior esforço quando realizadas a partir de um ambiente convencional de desenvolvimento de *software*, como o *C++ Builder*.

Na seqüência são apresentadas as demais opções construídas durante a implementação do módulo de reconhecimento sobre plataforma x86, bem como alguns resultados apurados a partir do mesmo. Por conta da posterior realização do treinamento a partir do ambiente MATLAB, também são alvo de discussão a forma como a mesma foi realizada e os resultados obtidos a partir daí.

5.2.3.1 Configuração

A opção de configuração da RNA, ilustrada na Figura 5.14, foi construída com o intuito principal de possibilitar a atribuição de padrões de saída distintos para cada uma das classes existentes nas redes de letras e números.

Uma das possíveis formas de codificar os padrões de saída (classes) consiste na utilização do esquema *um-para-M* (HAYKIN, 1999), onde apenas um dos *m* elementos que constituem o vetor que identifica cada classe é igual a “1”. Assim sendo, conforme ilustrado pelo item “B” na Figura 5.14, a RNA de letras é capaz de identificar 26 classes distintas

(A..Z), onde cada classe é definida por um vetor de 26 elementos. Cada um destes vetores contém o valor “1” em uma posição distinta dos demais, sendo o conteúdo dos demais elementos igual a “-1” (por conta do uso da função de ativação tangente hiperbólica). A mesma idéia se estende ao caso da RNA dos números, cuja codificação dos padrões de saída é ilustrada pelo item “C” na Figura 5.14. A única diferença diz respeito ao número de classes e o tamanho do vetor que as identifica, ambos igual a dez (0..9).

Figura 5.14 – Interface de configuração da RNA.

5.2.3.2 Geração do arquivo de padrões

A opção de geração do arquivo de padrões, ilustrada na Figura 5.15, foi implementada no intuito de automatizar o processo de geração de conjuntos de entradas utilizados no treinamento e na avaliação de desempenho das RNAs. A partir da especificação dos padrões de entrada desejados e do número de amostras por padrão, uma busca é realizada nos arquivos de origem (Figura 5.8(b)) com a intenção inicial de verificar a existência de amostras suficientes para cada padrão solicitado. Quando o resultado da busca é positivo, o arquivo de padrões é então gerado.

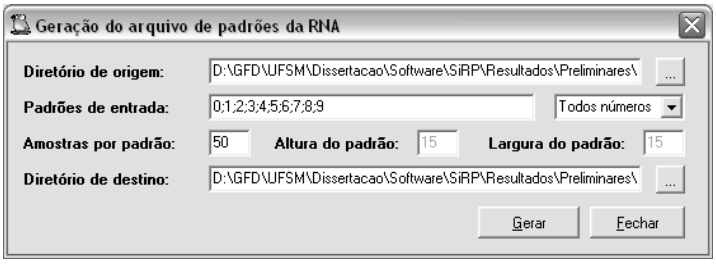


Figura 5.15 – Interface construída para permitir a geração dos arquivos contendo os padrões de entrada para o treinamento da RNA.

Durante a geração do arquivo, cada um dos padrões previamente selecionados é inicialmente organizado de uma forma a possibilitar a sua atribuição à camada de entrada da RNA. Como no arquivo de origem o padrão encontra-se armazenado na forma de uma matriz de *m* linhas e *n* colunas, o mesmo é mapeado para um vetor de *m* elementos pela concatenação de suas linhas, conforme ilustrado na Figura 5.16. Na seqüência, o padrão de saída relativo ao caractere representado pelo conteúdo do vetor é agregado ao mesmo, originando assim um registro no arquivo de padrões.

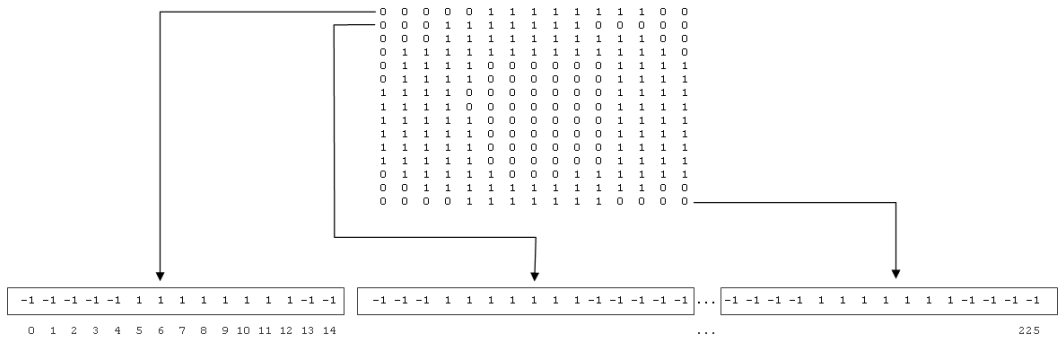


Figura 5.16 – Ilustração do processo de mapeamento da matriz de *bits* utilizada na representação dos caracteres para o vetor de entrada da RNA.

Ao término do processo tem-se um arquivo como o ilustrado pela Figura 5.17, onde cada linha representa um registro composto pelo padrão de entrada (item “A”) e seu respectivo padrão de saída (item “B”). No que diz respeito à ordenação dos registros, a mesma se dá primeiramente em função do número da amostra e em seguida pelo padrão. Neste sentido, os registros indicados pelo item “C” constituem a primeira amostra de cada um dos padrões.

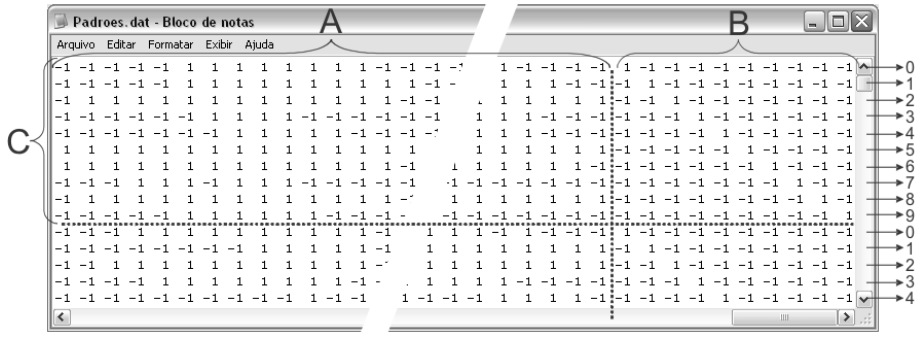


Figura 5.17 – Ilustração do conteúdo de um arquivo de padrões utilizado no treinamento e avaliação de RNAs de números.

5.2.3.3 Resultados preliminares a partir do módulo sobre plataforma x86

A partir do processo de treinamento e avaliação do desempenho de RNAs discutido em detalhes na Seção 5.2.3, deu-se a apuração dos resultados parciais ilustrados na Figura 5.18 para uma série de topologias de RNAs de números.

Em todos os casos, o número máximo de iterações utilizado foi igual a 300.000, sendo que na maioria das vezes o treinamento foi encerrado por conta do número máximo de iterações ter sido atingido. Entretanto, também na maioria das vezes, o EQM era praticamente igual ao especificado (0,03). Conforme mencionado na Seção 5.2.3, o modo de treinamento utilizado foi o seqüencial. A ordem de apresentação dos padrões foi a mesma utilizada pelo arquivo de padrões (Figura 5.17) para o caso ilustrado pela Figura 5.18(a), enquanto que para os demais casos os padrões foram apresentados de forma randômica.

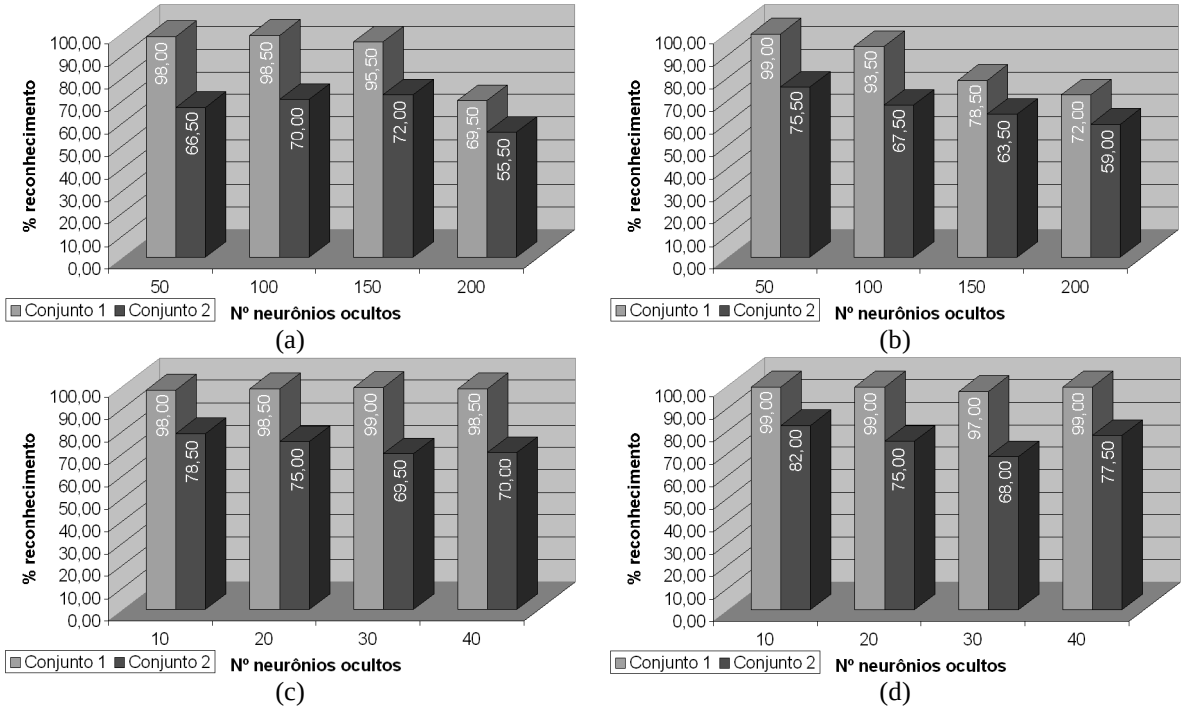


Figura 5.18 – Resultados parciais apurados para RNAs de números a partir do módulo construído sobre plataforma x86.

No que diz respeito às topologias utilizadas, em todas as situações foi utilizada uma camada escondida, exceto no caso ilustrado na Figura 5.18(d), onde foram utilizadas duas. Em relação ao número de neurônios nas camadas escondidas, o mesmo é ilustrado no eixo *x* de cada gráfico.

Cada par de resultados (Conjunto1, Conjunto 2) representa o maior percentual de reconhecimento obtido a partir de cinco treinamentos distintos para uma dada topologia. Neste sentido, as entradas que compõem o Conjunto 1 são as mesmas utilizadas para o treinamento da rede, enquanto os dados contidos no Conjunto 2 representam entradas não apresentadas anteriormente à mesma. Desta forma, era esperado que o desempenho do Conjunto 1 fosse sempre superior ao obtido pelo Conjunto 2. Mais do que isto, o percentual de reconhecimento do primeiro conjunto deveria ser igual ou ao menos próximo de 100%, haja vista que as redes foram treinadas a partir do mesmo. Nos casos onde este desempenho não foi alcançado, o resultado igualmente se refletiu no outro conjunto.

A partir da análise conjunta dos resultados, nota-se um melhor desempenho das redes onde foi utilizado um número menor de neurônios escondidos (Figura 5.18(c) e (d)), resultado este talvez relacionado ao número limitado de iterações. Já o fato das entradas terem sido apresentadas de forma randômica parece ter contribuído para o aumento do percentual de reconhecimento, se comparados o gráfico da Figura 5.18(a) e da Figura 5.18(b). Da mesma maneira, a utilização de duas camadas escondidas parece ter contribuído para uma ligeira melhora dos resultados (Figura 5.18(d) em relação à (c)).

5.2.3.4 Processo de avaliação e resultados preliminares a partir do ambiente MATLAB

Conforme mencionado no final da Seção 5.2.3, decorrido algum tempo após a construção da opção de treinamento/avaliação de RNAs sobre plataforma x86, tomou-se conhecimento e optou-se pela utilização do MATLAB para a realização desta tarefa. O MATLAB é um ambiente próprio à realização de simulações, fornecendo suporte a uma vasta gama de áreas, dentre as quais, redes neurais artificiais. A construção dos protótipos pode ser efetuada tanto de modo gráfico, como textual. Optou-se pela utilização do modo textual por conta da maior flexibilidade fornecida pelo mesmo. A construção de um protótipo a partir do modo textual é efetuada por intermédio de uma linguagem de programação proprietária, similar à Pascal. Neste sentido, geralmente o maior esforço se concentra na estruturação de rotinas pré-existentes no intuito de obter o resultado desejado. A agilidade na construção e validação de protótipos reside justamente no fato da existência deste vasto conjunto de algoritmos previamente validados.

No que diz respeito ao processo de treinamento e avaliação de desempenho de RNAs desenvolvido no ambiente MATLAB, o mesmo pode ser ilustrado pelo pseudocódigo abaixo. As instruções em itálico consistem em rotinas disponibilizadas pelo ambiente.

```
Crie lista de funções de treinamento;
Crie lista de nº de amostras por padrão;
Crie lista de topologias;

Para cada item na lista de funções de treinamento faça
        Para cada item na lista de nº de amostras por padrão faça
                Para cada item na lista de topologias faça
                        Leia arquivo de padrões de entrada(nº de amostras por padrão);
                        Crie rede neural artificial(função de treinamento);
                        Ajuste parâmetros de treinamento;
                        Para x de 1 até 10
                                Execute treinamento(conjunto padrões 1);
                                Execute simulação(conjunto padrões 1, conjunto padrões 2);
                                Apure desempenho da rede(conjunto padrões 1, conjunto padrões 2);
                                Armazene dados de treinamento e simulação em formato proprietário;
                                Armazene resultados de desempenho em arquivo Excel;
                        Fim Para;
                Fim Para;
        Fim Para;
Fim Para;
```

Inicialmente são criadas listas contendo diversas opções de funções de treinamento, número de amostras por padrão e topologias de rede, possibilitando assim a avaliação de diversas combinações das mesmas. No que diz respeito ao conteúdo de tais listas, a referente às funções de treinamento contém os seguintes algoritmos:

- Retropropagação de erro por descida do gradiente com taxa de aprendizado adaptativa (*traingda*): consiste no mesmo algoritmo discutido na Seção 5.2.3, com exceção do controle da variação da taxa de aprendizado, que é feito de forma automática neste caso;
- Retropropagação de erro por descida do gradiente com termo de momento (*traingdm*): consiste na versão do algoritmo de retropropagação que considera o termo de momento, mas que utiliza uma taxa de aprendizado fixa;
- Retropropagação de erro por descida do gradiente com taxa de aprendizado adaptativa e termo de momento (*traingdx*): consiste na consolidação dos algoritmos citados nos dois itens anteriores;
- Retropropagação de erro resiliente (*trainrp*): consiste num algoritmo que elimina o problema gerado pelo uso de entradas não normalizadas e funções sigmoidais, levando em consideração apenas o sinal da derivada parcial no cálculo do valor utilizado na atualização dos pesos;

- Retropropagação de erro por gradiente conjugado com atualizações Polak-Ribiére (*traincgp*): algoritmos de gradiente conjugado procuram acelerar o processo de convergência pela realização de uma busca ao longo de direções conjugadas , ao invés de apenas na direção da descida mais íngreme (método padrão);
- Retropropagação de erro por Levenberg-Marquardt (*trainlm*): consiste numa alternativa aos métodos de gradiente conjugado, baseada no emprego de um método *quasi*-Newton, onde a matriz de Hessian é aproximada por meio da matriz Jacobiana;
- Retropropagação de erro por gradiente conjugado escalado (*trainscg*): consiste num algoritmo que combina as características de *traincgp* e *trainlm*, no intuito de acelerar ainda mais o processo de convergência;
- Retropropagação de erro por regularização Bayesiana (*trainbr*): consiste num algoritmo que procura melhorar a capacidade de generalização da RNA a partir do emprego do método de regularização, o qual emprega uma função de custo modificada.

Já no tocante à lista de número de amostras por padrão, foram efetuados treinamentos utilizando 15, 20 e 25 amostras por padrão, cujo objetivo foi o de descobrir a influência deste parâmetro no aprendizado da rede. Finalmente, em relação à lista de topologias, foram avaliadas redes com duas, três e quatro camadas (uma, duas e três camadas escondidas, respectivamente), onde o número de neurônios escondidos em cada camada variou de 20 a 100, em intervalos de 20 unidades.

Na seqüência, 10 rodadas do processo de treinamento e avaliação são executadas para cada uma das combinações possíveis dos elementos contidos na listas. Neste sentido, o primeiro passo consiste na leitura do arquivo de padrões idêntico ao utilizado no processo construído sobre plataforma x86. No passo seguinte é criada a rede neural PMC levando em consideração o algoritmo de treinamento atualmente selecionado, sendo em seguida efetuado o ajuste dos parâmetros de treinamento da mesma. Num primeiro momento foram mantidos os valores padrão dos parâmetros, sendo alguns deles alterados posteriormente, conforme ilustrado na Tabela 5.1, em função dos resultados obtidos a partir de testes iniciais.

Da mesma forma que anteriormente discutido, o treinamento se estende até o momento que um dos critérios de parada é alcançado, sendo que neste caso os critérios consistem nas três primeiras entradas da Tabela 5.1. No que diz respeito ao modo de treinamento, a

atualização dos pesos foi realizada *em lote*, haja vista que este é o modo padrão neste caso. Findada a etapa de treinamento, os mesmos procedimentos necessários à apuração do desempenho da rede previamente mencionados são executados. Neste sentido, a principal diferença entre o processo anterior (plataforma x86) e o atual (ambiente MATLAB) reside tanto na quantidade, quanto no modo de armazenamento dos resultados apurados. Enquanto anteriormente, no que se refere à simulação em si, apenas os pesos das sinapses eram armazenados, neste caso foi possível o armazenamento de estruturas completas contendo dados da rede, do treinamento e das simulações a partir da chamada de uma única rotina. Já em relação aos resultados da simulação, que anteriormente se resumiam a três informações, agora contemplam dados como topologia, número de iterações, desempenho e erros por classe. Tais dados foram armazenados no formato de planilha eletrônica, o que facilitou a posterior avaliação da massa de resultados apurados.

Tabela 5.1 – Novos valores atribuídos aos algoritmos de treinamento no ambiente MATLAB.

Parâmetro	Algoritmo	Valor
Número máximo de épocas para treinamento	Todos	10.000
Tempo máximo de treinamento	Todos	120 (s)
Meta de desempenho	Todos	$1x10^{-9}$
Taxa de aprendizado	*traingda*, *traingdm*, *traingdx*, *trainrp*	0,025
Razão de incremento da taxa de aprendizado	*traingda*, *traingdx*	1,15
Constante de momento	*traingdm*, *traingdx*	0,8

A nova rodada de avaliações foi realizada a partir de um segundo conjunto de imagens, composto por 3.000 exemplares. Já que o processo discutido na Seção 5.2.2, teve de ser executado novamente, uma nova implementação que permitiu a classificação da qualidade da placa na imagem foi realizada no intuito de permitir uma correta apuração do resultado final. Tal classificação é discutida no capítulo referente aos resultados do trabalho.

Dos oito algoritmos de treinamento inicialmente selecionados, três foram descartados (*traingdm*, *trainbr*, *trainlm*) por conta da demora excessiva do processo de treinamento de avaliação. Considerando uma RNA de números de topologia 225x20x10 e a utilização de 15 amostras por padrão, uma única rodada do processo levava pelo menos 80 segundos para ser executada quando do uso dos três algoritmos descartados. Já no caso da utilização dos demais algoritmos, a mesma rodada demorava em média três segundos. Como o número de combinações possível para cada algoritmo era de 4650 (três opções de número de amostras por padrão x 155 opções de topologia x 10 rodadas), decidiu-se não utilizar algoritmos onde a mais simples topologia avaliada já consumisse tanto tempo.

Com base nas considerações discutidas nos parágrafos anteriores, foi então realizado o processo de treinamento e avaliação a partir do ambiente MATLAB, sendo os resultados iniciais ilustrados na Figura 5.19 e na Figura 5.20. Tais resultados foram apurados considerando o emprego de padrões de entrada não utilizados em nenhum momento durante o treinamento.

Os resultados exibidos na Figura 5.19 referem-se à RNA de letras, onde os gráficos (a), (b) e (c) ilustram os máximos percentuais de reconhecimento obtidos para todas as

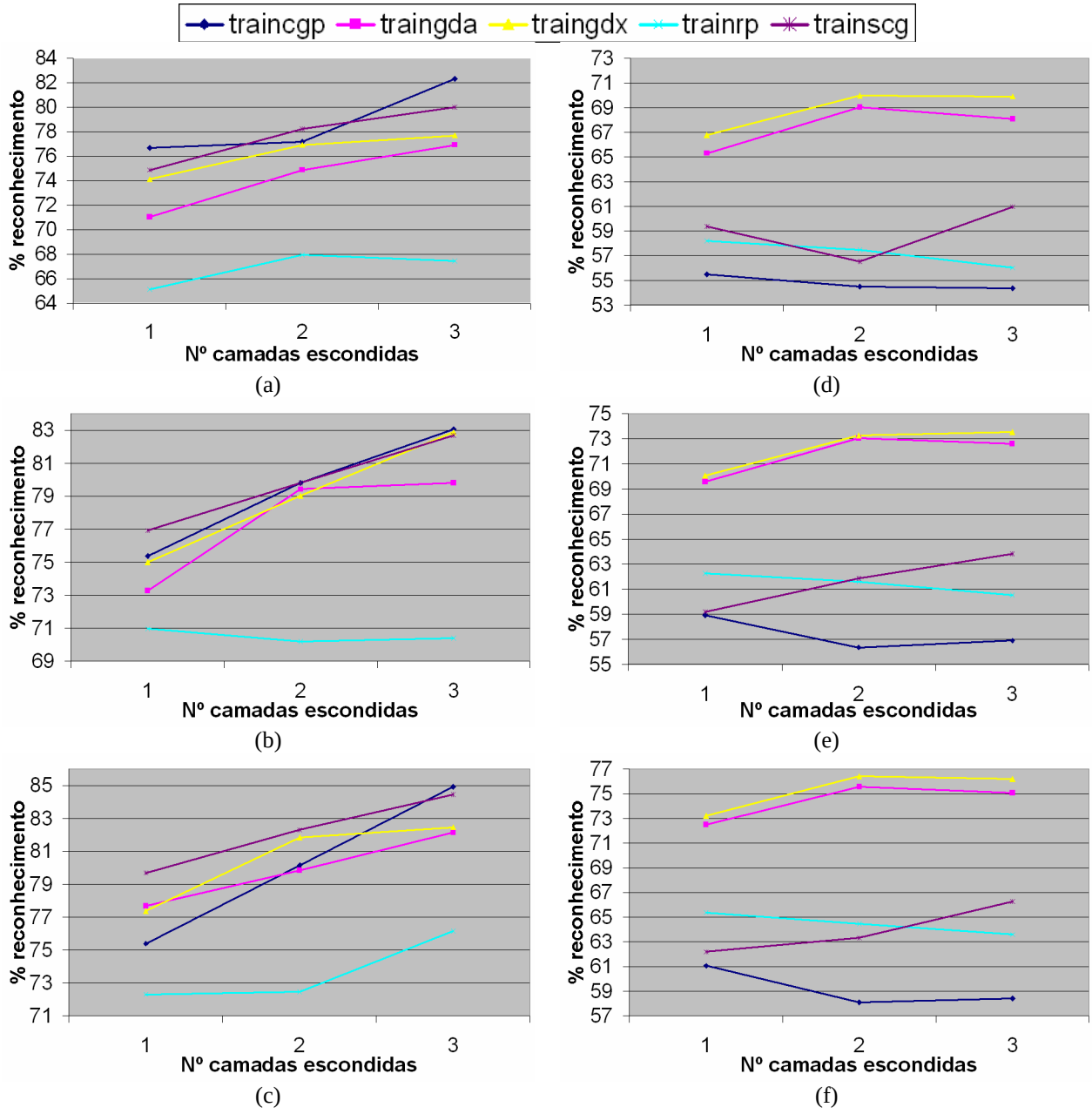


Figura 5.19 – Percentuais (a)-(c) máximos e (d)-(f) médios de reconhecimento apurados para as RNAs de letras nas simulações conduzidas no ambiente MATLAB, considerando a utilização (a),(d) 15 , (b),(e) 20 e (c),(f) 25 amostras por padrão na etapa de treinamento.

topologias avaliadas, considerando a utilização de 15, 20 e 25 amostras por padrão durante o treinamento, respectivamente. A partir da análise de tais gráficos, nota-se um melhor desempenho por parte dos algoritmos *traincgp* e *trainscg* na maior parte dos casos. Entretanto, quando a mesma análise é realizada levando em consideração o percentual de reconhecimento médio (das 10 rodadas realizadas para cada algoritmo), os resultados são diferentes, conforme ilustrado pelos gráficos (d), (e) e (f) na mesma figura. Neste caso os

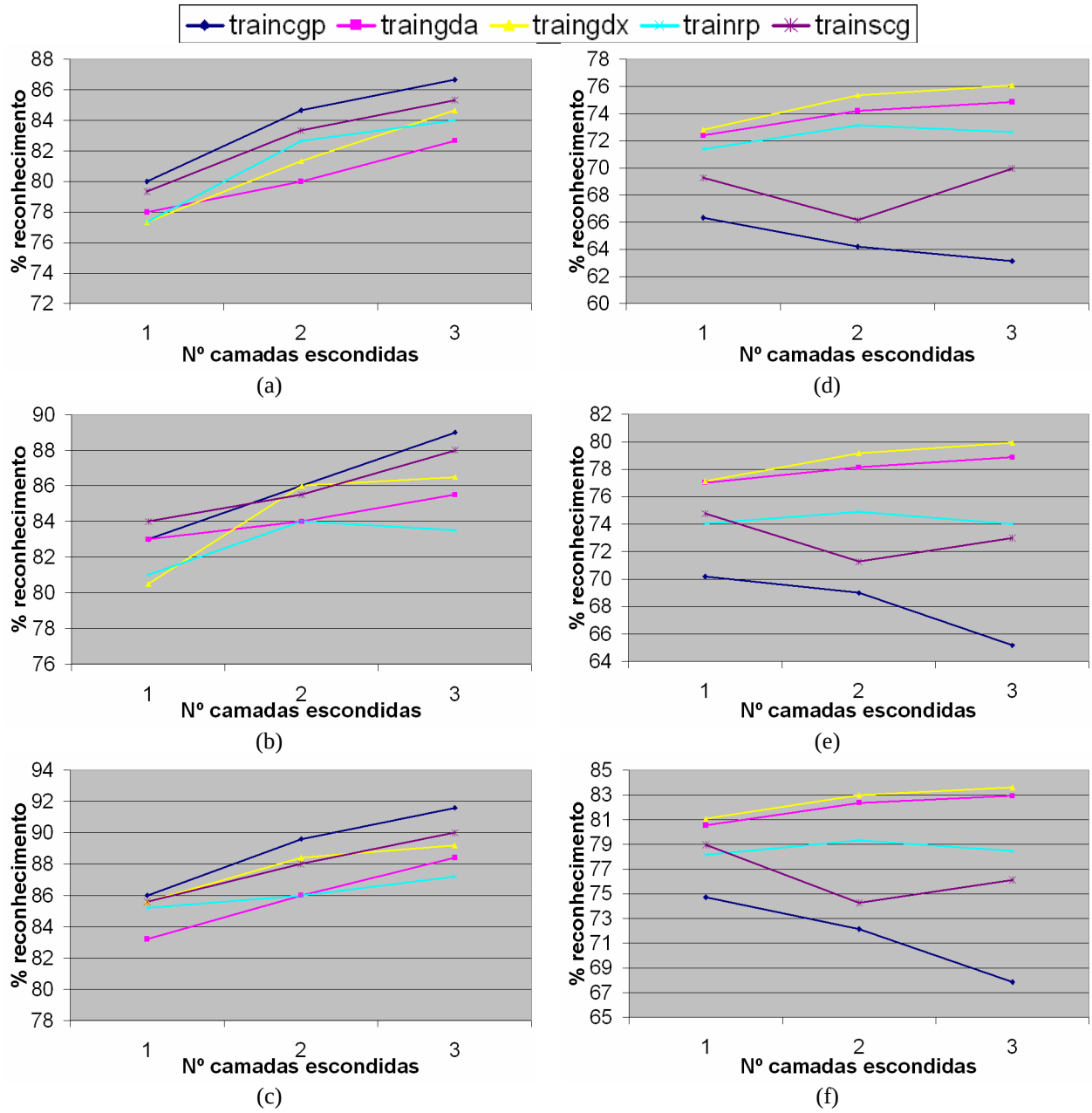


Figura 5.20 – Percentuais (a)-(c) máximos e (d)-(f) médios de reconhecimento apurados para as RNAs de números nas simulações conduzidas no ambiente MATLAB, considerando a utilização (a),(d) 15 , (b),(e) 20 e (c),(f) 25 amostras por padrão na etapa de treinamento.

algoritmos *traingdx* e *traingda* se sobressaem, fato este que pode indicar uma falta de consistência dos resultados exibidos em (a), (b) e (c).

O mesmo comportamento apresentado pela RNA de letras torna a ocorrer no caso da RNA de números, conforme ilustram os resultados na Figura 5.20. Desta maneira, decidiu-se selecionar ambas as funções de treinamento *traingdx* e *traincgp* para uma posterior avaliação de desempenho em nível de placa (os sete caracteres), haja vista que os percentuais aqui apresentados se referem ao reconhecimento dos caracteres de forma isolada.

Entretanto, como tal avaliação em nível de placa será realizada a partir do sistema sobre plataforma x86, faz-se necessária a especificação não apenas da função de treinamento, mas também da topologia de rede a ser utilizada. Levando em consideração os resultados exibidos na Figura 5.19 e na Figura 5.20, nota-se que em ambos os casos o uso de duas camadas escondidas proporcionou o melhor custo-benefício. A partir de tal constatação, foram recuperados os resultados apurados para as funções *traingdx* e *traincgp*, onde o número de camadas escondidas era igual a duas, estando tais resultados ilustrados na Figura 5.21 e na Figura 5.22, respectivamente. Considerando tais resultados, a escolha inicial de cada

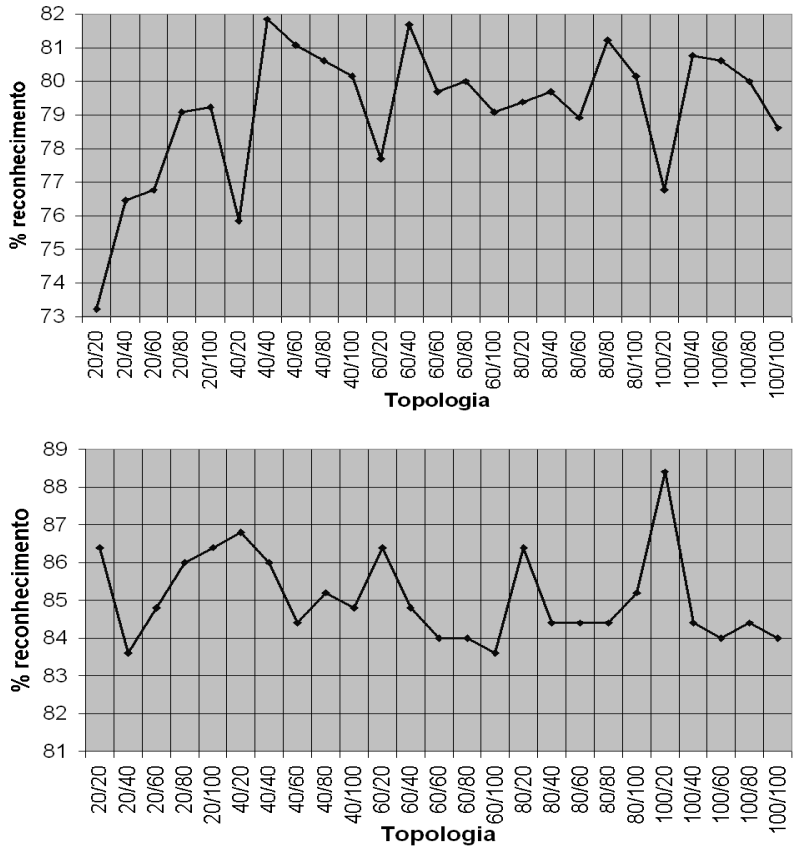
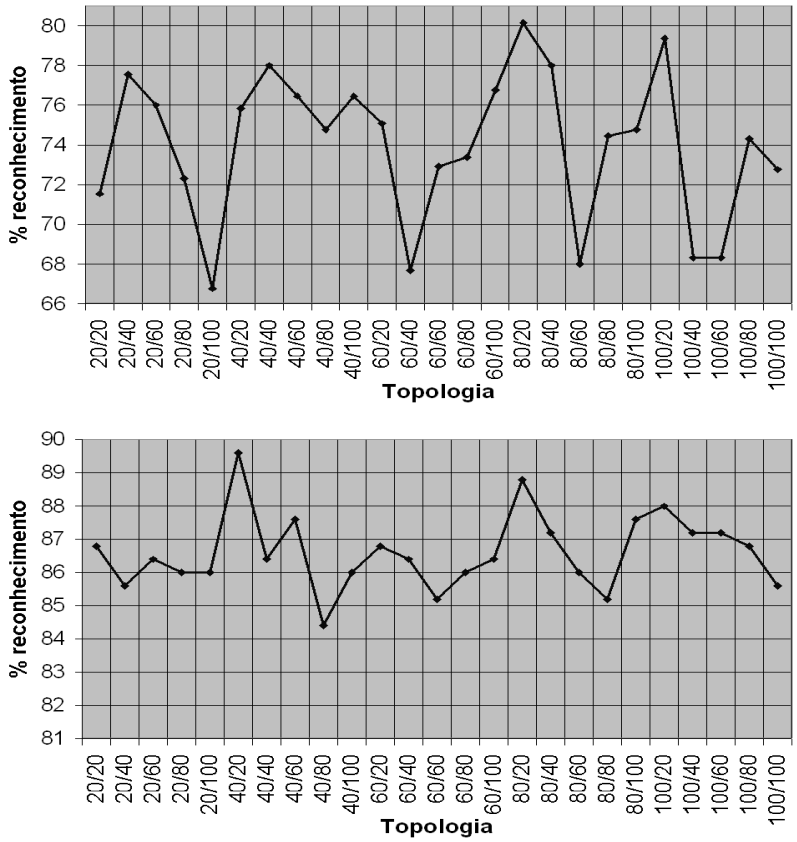


Figura 5.21 – Percentuais máximos de reconhecimento apurados para redes de (a) letras e (b) números em função do número de neurônios em cada uma das duas camadas escondidas, a partir da utilização do algoritmo de treinamento *traingdx*.

topologia foi baseada no critério de que a mesma deveria ser idêntica para ambas funções de treinamento e ao mesmo tempo apresentar um elevado custo-benefício. Assim sendo, para o caso da RNA de letras (Figura 5.21(a) para a função *traingdx* e Figura 5.22(a) para *traincgp*), a topologia inicialmente selecionada foi a 40x40. Já no caso da RNA de números (Figura 5.21(b) para a função *traingdx* e Figura 5.22(b) para *traincgp*), a topologia inicialmente selecionada foi a 20x20.



(a)

(b)

Figura 5.22 – Percentuais máximos de reconhecimento apurados para redes de (a) letras e (b) números em função do número de neurônios em cada uma das duas camadas escondidas, a partir da utilização do algoritmo de treinamento *traincgp*.

Na seqüência, é descrita a implementação do sistema sobre plataforma embarcada.

5.3 Sistema sobre plataforma embarcada

5.3.1 Introdução

Ao término da implementação da versão em *software* sobre plataforma x86 do sistema, processo este que serviu também à seleção dos módulos a serem utilizados na construção da versão embarcada, deu-se início a implementação desta última.

Inicialmente foi realizada a construção do bloco de *hardware*, tendo sido o mesmo primeiramente validado em nível de transferência de registradores, posteriormente em nível de portas lógicas e finalmente em nível físico. Tal processo de implementação e validação foi executado de forma distinta para cada um dos módulos que compõem o bloco, haja vista que uma abordagem hierárquica facilita o processo como um todo.

No que diz respeito às ferramentas utilizadas durante o processo, foram empregados tanto *softwares* comerciais, quanto um desenvolvido pelo autor, bem como um analisador lógico no caso da depuração de casos pontuais. Na etapa de projeto foi empregada a ferramenta *HDL Designer*, versão 2006.1, da empresa *Mentor Graphics*. Já nas etapas de síntese e validação em níveis de transferência de registradores e portas lógicas, foi empregada a ferramenta *Integrated Software Environment* (ISE), versão 8.2.03i, fornecida pela *Xilinx Incorporated*. A partir do ambiente ISE, optou-se pela utilização do sintetizador padrão (*Xilinx Synthesis Tool* (XST)) e pelo emprego do simulador *ModelSim*, versão 6.2d, este último também fornecido pela *Mentor Graphics*, mas com integração ao ambiente ISE. Finalmente, a validação física foi realizada a partir de um *software* construído pelo autor sobre plataforma x86, o qual é discutido na Seção 6.2.3. Também nesta etapa foi empregado o analisador lógico de 64 canais, modelo 1682AD, fabricado pela *Agilent Technologies*, em situações específicas de depuração do circuito implementado sobre o FPGA. A não utilização do recurso de co-simulação justifica-se tanto pela complexidade da operação, quanto por uma das razões já descrita em Ou (2005): o longo tempo consumido pela mesma.

Finalizada a construção do bloco de *hardware*, partiu-se então para a implementação do sistema embarcado em si, processo esse realizado a partir da ferramenta *Embedded Development Kit* (EDK), versão 8.2.02i, fornecido pela empresa *Xilinx Incorporated*. Neste sentido, o primeiro passo consistiu na definição da plataforma básica de *hardware* e *software* sobre a qual o sistema seria implementado. Estando o sistema base disponível, o próximo passo consistiu na implementação da RNA sobre a plataforma embarcada. O módulo de reconhecimento foi então implementado e posteriormente validado diretamente sobre a plataforma embarcada.

Finalmente, a última etapa da implementação consistiu na consolidação dos blocos e *hardware* e *software* sobre a plataforma embarcada, onde o bloco de *hardware* foi empregado como um acelerador de aplicação, conforme previamente ilustrada na Figura 3.3.

Nas próximas seções deste capítulo são discutidos em detalhes aspectos como o conteúdo do bloco de *hardware*, a implementação do bloco de *software*, bem como a conexão dos dois sobre a plataforma embarcada.

5.3.2 Bloco de *hardware*

5.3.2.1 Introdução

O bloco de *hardware* é composto por um conjunto de módulos básicos, três módulos principais (FPLA, OCE e ON), além de outros módulos secundários de comunicação, armazenamento e geração de *clocks*. Dentre as tarefas executadas pelo conjunto de módulos básicos, a principal consiste em coordenar a interação entre os demais módulos, bem como interpretar, executar e retornar respostas aos comandos enviados pela interface de *software* sobre plataforma x86. Já o módulo FPLA é responsável pela tarefa de etiquetação de componentes conectados, enquanto o módulo OCE executa a extração das coordenadas dos objetos detectados pelo módulo FPLA. Finalmente, o módulo ON realiza a normalização dos objetos classificados como caracteres.

A seguir, cada um dos módulos é discutido em detalhes.

5.3.2.2 O conjunto de módulos básicos

No intuito de prover funções de controle, comunicação e armazenamento ao bloco de *hardware*, bem como possibilitar a validação em nível físico dos demais módulos, inicialmente o conjunto de blocos ilustrado na Figura 5.23 foi construído e validado. Tal conjunto é composto pela unidade de controle central (SIRP_CP), a de comunicação serial (*Universal Asynchronous Receiver-Transmitter* – UART), bem como as duas de armazenamento (RAM_BIN e RAM_SYM) e seus respectivos multiplexadores (RAM_BIN_MUX e RAM_SYM_MUX). Além disso, também conta com um circuito de geração de *clock*, que deriva freqüências intermediárias a partir de um *clock* principal (*clk*) de 100 MHz. Já a inversão do sinal *rst* se deve ao fato da placa de prototipação utilizada (XUP-V2Pro) fornecer um sinal ativo em baixo para os botões de contato momentâneo.

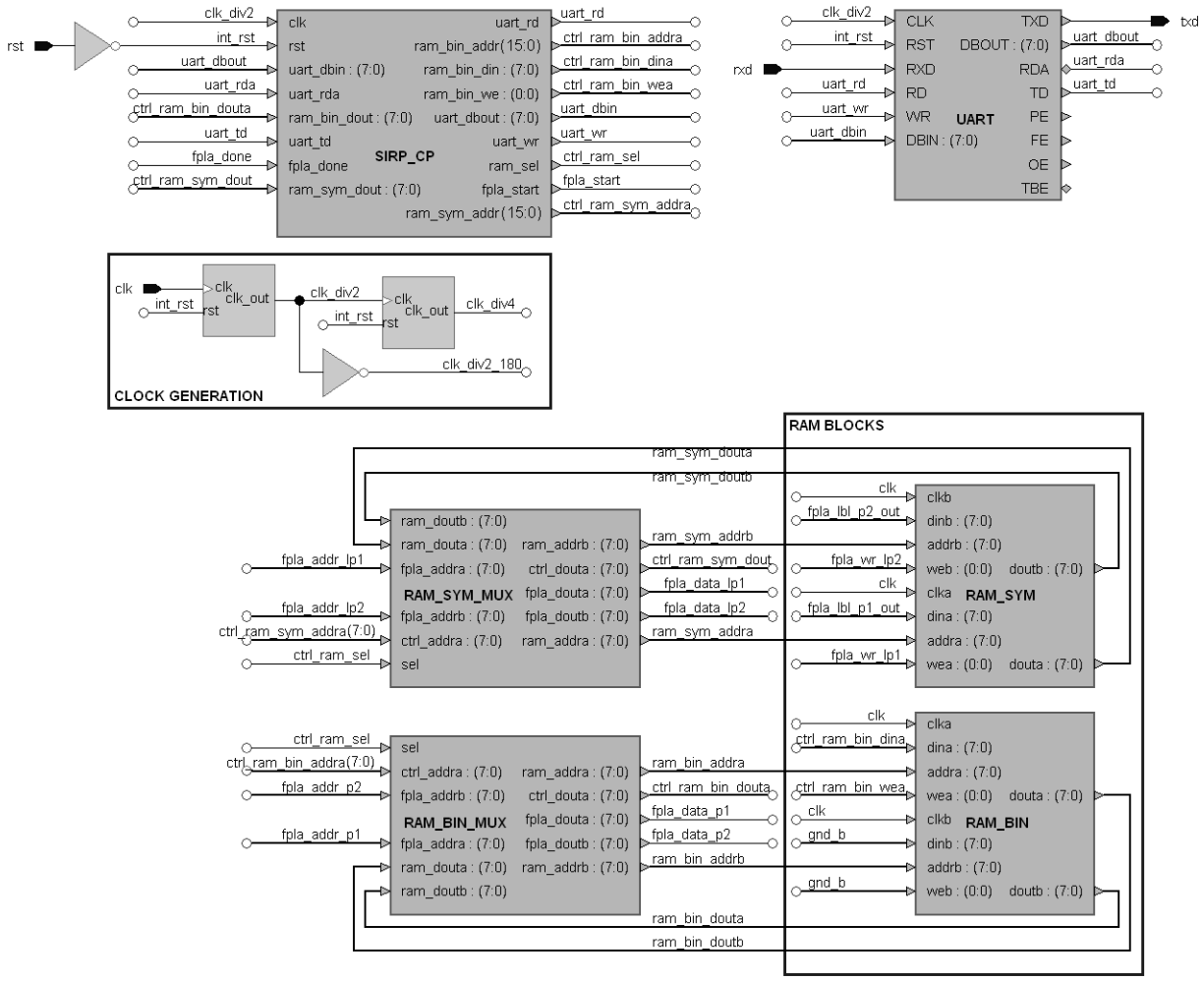


Figura 5.23 – Diagrama de blocos do conjunto de módulos básicos.

O módulo SIRP_CP é encarregado pelo controle central do bloco de *hardware*, sendo composto por uma máquina de estados principal e outra auxiliar, esta última responsável apenas pela simplificação do processo de envio de dados. Já a máquina principal, ilustrada na Figura 5.24, é encarregada de receber instruções da interface de *software* rodando sobre plataforma x86, providenciar sua execução, bem como retornar dados e o *status* de execução da mesma. Neste sentido, *resm_reset* constitui o estado inicial da máquina, executado apenas quando o sistema é posto em funcionamento, com a intenção de atribuir valores iniciais aos sinais envolvidos. No segundo ciclo de *clock* a máquina se encontra no estado *resm_idle*, onde permanece até que o sub-módulo UART sinalize a chegada de um *byte* (uart_rda = ‘1’). Quando tal evento ocorre, a máquina solicita a leitura do dado (uart_rd = ‘1’) e avança para o estado *resm_read_uart*, onde é realizada a carga do mesmo para um registrador interno (rcvd_data <= uart_dbin). Assim que o módulo UART indica a leitura com sucesso do dado (uart_rda = ‘0’), a máquina desfaz a solicitação de leitura (uart_rd = ‘0’) e avança para o

estado *resm_decode*. Neste momento o dado recebido é interpretado e máquina direciona o fluxo de execução de acordo com o conteúdo do registrador *cmd_code*. Caso o dado recém recebido consista no primeiro da seqüência que compõe um comando (cmd_code = CMD_NONE), a máquina retorna ao estado *resm_idle* para aguardar os demais *bytes*, caso contrário, segue para o estado referente ao comando.

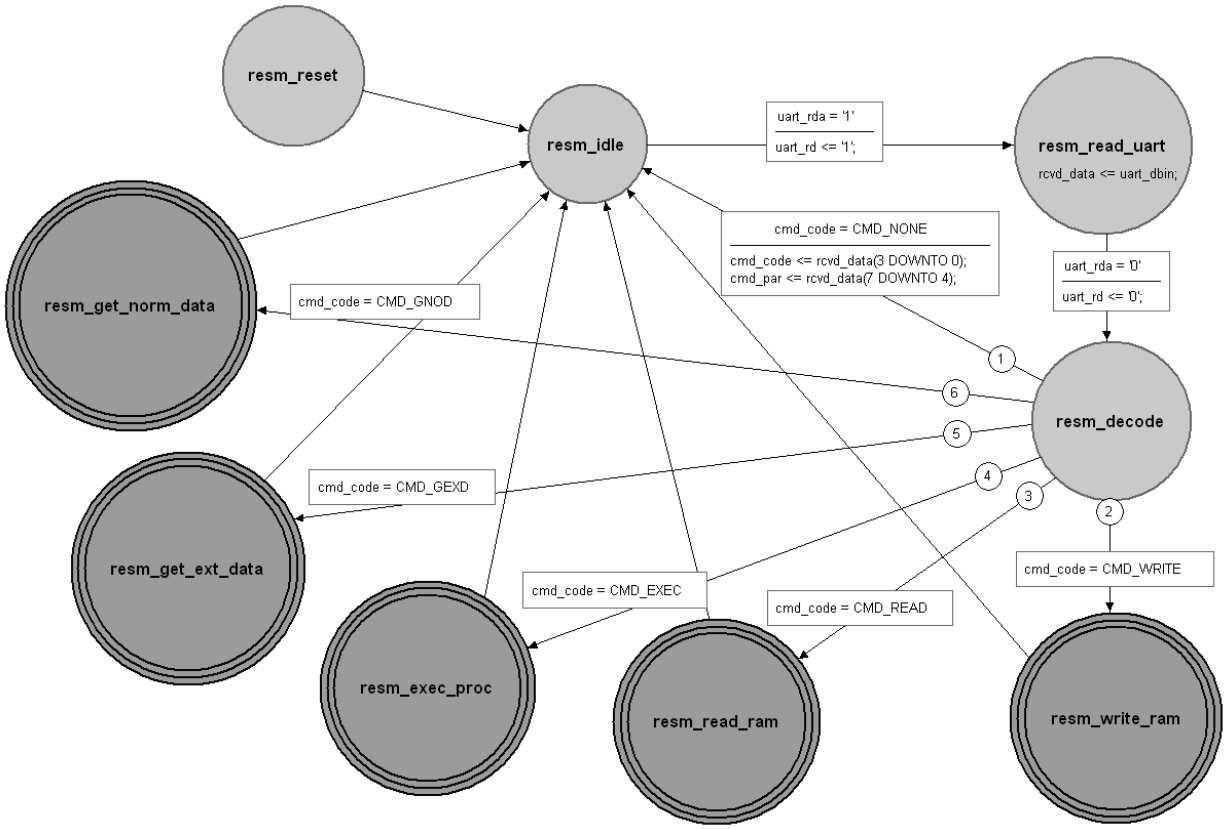


Figura 5.24 – Máquina de estados principal que constitui o módulo SIRP_CP.

Cada comando é composto por um ou mais *bytes*, sendo que o primeiro identifica o código do comando a ser executado, enquanto que os demais constituem parâmetros do mesmo. Entretanto, por conta da estrutura da máquina de estados, é necessário o envio de pelo menos dois *bytes* para que a o comando seja processado. Isto ocorre porque após o recebimento do *byte* de código, a máquina volta para o estado inicial à espera de parâmetros. Outro detalhe consiste na utilização do próprio *byte* de código na determinação da RAM de origem, no caso do comando CMD_READ. Como tal sinalização pode ser feita por meio de um único *bit*, optou-se por utilizar o de índice 5 no *byte* de código, ao invés de adicionar um *byte* inteiro ao comando. Neste caso, o valor ‘0’ seleciona a memória simbólica (RAM_SYM), enquanto o valor ‘1’, a binária (RAM_BIN). A Tabela 5.2 ilustra o tamanho de

cada comando implementado, bem como identifica o significado de cada *byte*. Apesar dos comandos CMD_WRITE e CMD_READ definirem endereços de dois *bytes*, inicialmente apenas um foi utilizado, conforme justificado em seguida, na discussão sobre os módulos de memória.

Tabela 5.2 – Tabela de comandos implementados no bloco de *hardware.*

Código	Descrição	Parâmetro 1	Parâmetro 2	Parâmetro 3
	1º *byte*	2º *byte*	3º *byte*	4º *byte*
CMD_WRITE	Escrita de dado em RAM	Endereço (parte baixa – *bits* 7:0)	Endereço (parte alta – *bits* 15:8)	Dado
CMD_READ	Leitura de dado de RAM	Endereço (parte baixa – *bits* 7:0)	Endereço (parte alta – *bits* 15:8)	
CMD_EXEC	Execução do processo de extração de caracteres	Indicador de envio do nº ciclos gastos na execução do processo		
CMD_GEXD	Leitura dos objetos encontrados	Índice do objeto na lista de coordenadas		
CMD_GNOD	Leitura da RAM contendo todos os caracteres normalizados			

Cada um dos estados responsáveis pela execução de um comando (*resm_write_ram*, *resm_read_ram*, *resm_exec_proc*, *resm_get_ext_data* e *resm_get_norm_data*), encontra-se ilustrado na Figura 5.24 como um *estado hierárquico*. O conceito de estado hierárquico é proveniente da ferramenta *HDL Desinger*, que o emprega com a intenção de aumentar o nível de abstração e facilitar o entendimento da máquina de estados como um todo. Neste sentido, cada estado hierárquico é composto por uma série de outros estados necessários à execução de uma tarefa, mas que aqui foram abstraídos para favorecer o entendimento da máquina de estados principal. Tal abstração é realizada apenas em nível gráfico, não se refletindo na descrição *Hardware Description Language* (HDL) da máquina de estados.

No que diz respeito ao módulo UART, o mesmo foi implementado a partir do módulo HDL fornecido por Digilent (2006). Tal componente se encarrega da transmissão e recepção de palavras de oito *bits* de forma serial, conforme a taxa de transmissão especificada, que neste caso é de 38.400 *bits* por segundo. As palavras a serem enviadas devem ser informadas na porta *DBIN*, enquanto as recebidas são disponibilizadas na porta *DBOUT*. Sinais de indicação de erros de recepção são disponibilizados a partir da portas *PE*, *FE* e *OE*, entretanto nenhum deles foi utilizado nesta implementação. A partir da implementação original, foi criada a porta *TD*, cuja função é a de indicar o término de uma operação de envio, o que também é sinalizado de uma forma não tão prática pela porta *TBE*.

Ambos os módulos RAM_BIN e RAM_SYM consistem em memórias voláteis de acesso duplo, construídas a partir de recursos próprios para este fim disponíveis no FPGA. A implementação dos mesmos se deu pela ferramenta *CORE Generator*, integrante do ambiente ISE, a partir da especificação de parâmetros como o tipo de memória (RAM/ROM, acesso simples/duplo), largura da palavra, número de palavras disponíveis, etc. Inicialmente foram utilizadas memórias de 256 palavras de oito *bits* cada para viabilizar os testes de validação, até porque não se tinha então a definição do tamanho máximo necessário. A principal diferença entre tais módulos reside no fato do intitulado RAM_BIN armazenar a imagem de entrada, no formato binário, enquanto o outro armazena a imagem simbólica resultante do processo de etiquetação de componentes conectados. As operações de escrita e leitura nos blocos de memória binária e simbólica são realizadas com auxílio de multiplexadores, haja vista que tais operações devem ser possibilitadas para ambos os módulos SIRP_CP e FPLA.

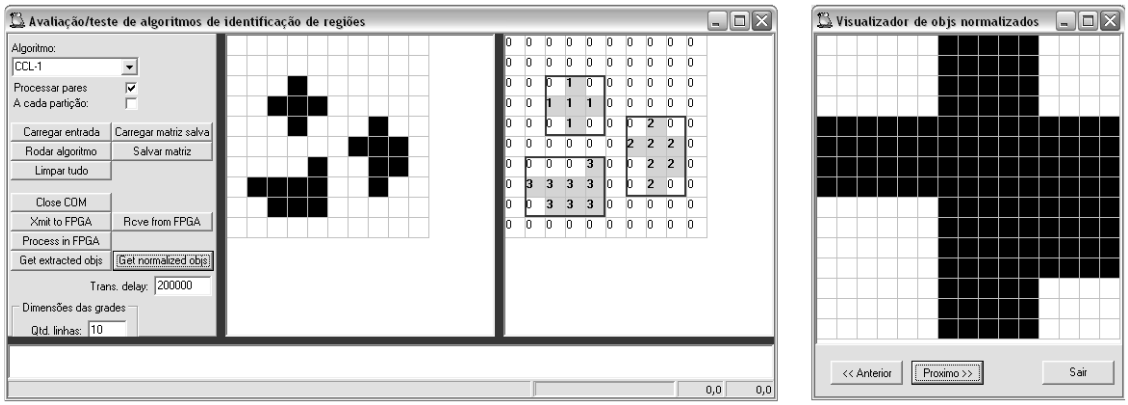
No caso da memória binária (RAM_BIN), as operações de escrita são efetuadas apenas a partir do módulo SIRP_CP, enquanto as de leitura podem ser realizadas também pelo módulo FPLA. Desta maneira, o multiplexador RAM_BIN_MUX é utilizado para selecionar o endereço de uma operação (portas *addra* e *addrb*) na memória binária, bem como redirecionar o conteúdo dos barramentos de saída (*douta* e *doutb*), quando tal operação for de leitura. Caso o módulo SIRP_CP esteja realizando uma operação de escrita ou leitura, o sinal *ctrl_ram_bin_addra* é o que define o endereço, enquanto o dado contido em tal endereço é redirecionado para o barramento *ctrl_ram_bin_douta*. Caso contrário, os endereços de leitura são definidos pelos sinais *fpla_addr_p1* e *fpla_addr_p2*, sendo os dados contidos em tais endereços redirecionados para os barramentos *fpla_data_p1* e *fpla_data_p2*, respectivmente. Como numa operação de escrita apenas 1 *byte* de dado é processado por vez, não há razão para utilizar o recurso de acesso duplo. Deste modo, tanto o barramento de entrada *dinb*, quanto o sinal de controle de escrita *web*, ambos situados no bloco RAM_BIN, foram desativados.

A mesma interação entre a memória binária e os blocos RAM_BIN_MUX, SIRP_CP e FPLA ocorre em relação à memória simbólica (RAM_SYM) e os blocos RAM_SYM_MUX, SIRP_CP e FPLA. A única diferença diz respeito ao fato de que neste caso o bloco SIRP_CP realiza apenas operações de leitura, enquanto o bloco FPLA executa ambas operações de leitura e escrita.

Simultaneamente à construção do conjunto de módulos básicos foi realizada a implementação da interface de *software* sobre plataforma x86 que permitiu a validação física dos módulos conforme os mesmos eram construídos. Tal interface originalmente se prestou à

validação de algoritmos de etiquetação de componentes conectados, ainda durante a construção da versão do sistema sobre plataforma x86. Como muitos recursos presentes nesta interface serviam ao propósito da validação dos módulos de hardware, optou-se por incrementar a mesma com as funcionalidades ainda necessárias ao invés de construir outra opção específica. Tais funcionalidades foram sendo implementadas conforme os blocos de *hardware* iam sendo construídos e validados primeiramente em nível de transferência de registradores e portas lógicas.

Como resultado, obteve-se a interface ilustrada na Figura 5.25(a), a qual permite a transmissão da matriz binária ao FPGA (botão "*Xmit to FPGA*"), a execução do processamento (botão "*Process in FPGA*"), bem como a apuração dos resultados. Neste sentido, é possível ler o conteúdo das memórias binária, simbólica (botão "*Rcve from FPGA*") e de caracteres normalizados (botão "*Get normalized objs*"), sendo o conteúdo desta última exibido numa janela à parte, ilustrada pela Figura 5.25(b). Também é possível a leitura da matriz de coordenadas (botão "*Get extracted objs*"), procedimento este que causa a delimitação dos objetos da matriz da direita na Figura 5.25(a) por retângulos vermelhos.



(a) (b)

Figura 5.25 – Ilustração da (a) interface de *software* sobre plataforma x86 utilizada na validação dos módulos de *harware*. No detalhe (b) é exibido um dos objetos normalizados.

5.3.2.3 Módulo FPLA

Finalizada a construção e validação do conjunto de módulos básicos, o foco voltou-se ao módulo FPLA, o qual é baseado na arquitetura proposta por Yang *et al.* (2005). Conforme ilustrado na Figura 5.26, o módulo é composto pelos sub-módulos FPLA_CP e FPLA_DP, bem como pelo bloco SEC_SCAN_EXT_GEN. Neste sentido, o sub-módulo FPLA_CP é o responsável pelo controle do processo de etiquetação de componentes conectados. Dentre as

operações executadas pelo mesmo destacam-se o endereçamento (*addr_p1*, *addr_p2*) da memória binária, endereçamento (*addr_lp1*, *addr_lp2*) e escrita (*wr_lp1*, *wr_lp2*) da memória simbólica, bem como a geração de outros sinais de controle (*busy*, *preset_cra*, *sec_scan*). Já o sub-módulo FPLA_DP é o encarregado pela tarefa de processamento em si, a partir dos valores de *pixels*, etiquetas e sinais de controle fornecidos pelo sub-módulo de controle. Finalmente, o bloco SEC_SCAN_EXT_GEN encarrega-se da geração do sinal *sec_scan_ext*, que constitui uma variante do sinal *sec_scan*, utilizado especificamente pelo módulo OCE.

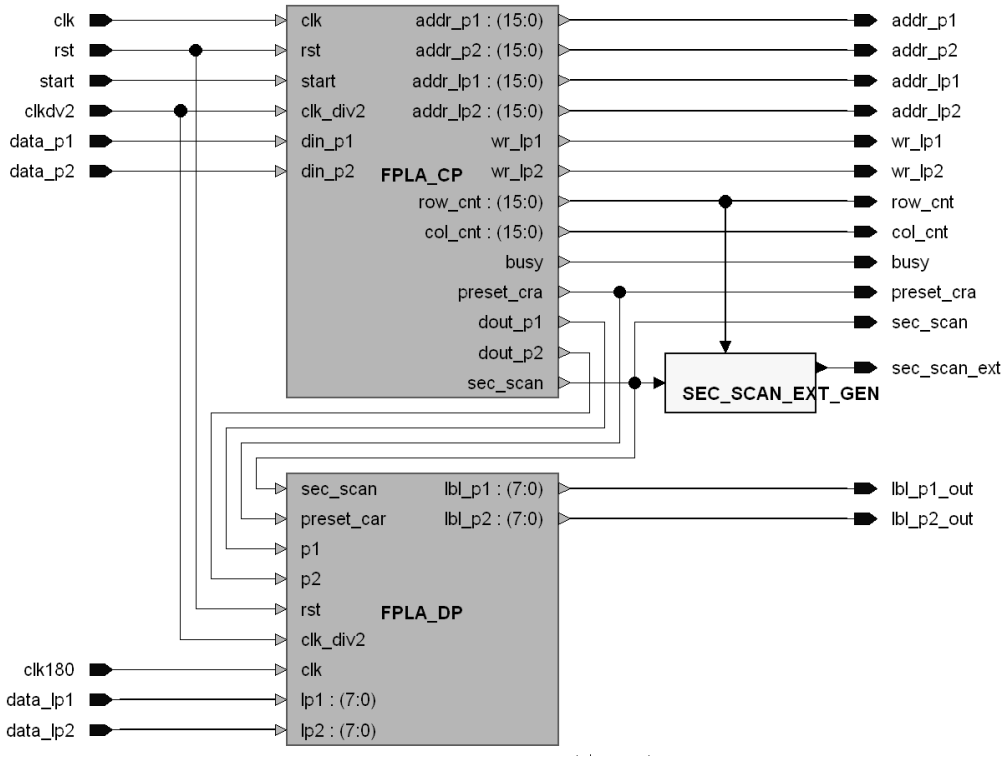


Figura 5.26 – Diagrama de blocos do módulo FPLA.

Descendo um nível na hierarquia do módulo FPLA, torna-se possível a visualização do conteúdo de seus sub-módulos. O primeiro deles, FPLA_CP, é constituído pela máquina de estados ilustrada na Figura 5.27. No que diz respeito ao funcionamento da mesma, inicialmente ela se encontra no estado *waiting*. No momento em que o bloco de controle central (SIRP_CP) solicita o início de operação (start = ‘1’), a máquina sinaliza que está em fase do processamento (busy <= ‘1’) e avança para o estado *sync*. Tal estado permanece como o atual até que ocorra a sincronização dos sinais *clk* e *clk_div2* (slow_clk_level = ‘0’), o que é imprescindível para o correto fornecimento dos *pixels* ao sub-módulo FPLA_DP. Quando ocorre a sincronização, a máquina sinaliza o reinício de estruturas internas ao sub-módulo FPLA_DP e avança para o estado *address*. Deste momento em diante a máquina executa a

primeira etapa do processamento, alternando entre os estados *address* e *feed* enquanto hajam *pixels* a serem processados na memória binária (int_col_cnt <= cols). Terminada a primeira etapa, a máquina sinaliza o início da segunda e avança para o estado *get_label*. De maneira similar à primeira etapa, a máquina alterna entre os estados *get_label* e *set_label* enquanto hajam dados a serem processados, entretanto agora levando em consideração a memória simbólica. Ao término da segunda etapa, a máquina volta ao estado inicial (*waiting*) e ajusta o valor dos sinais de controle.

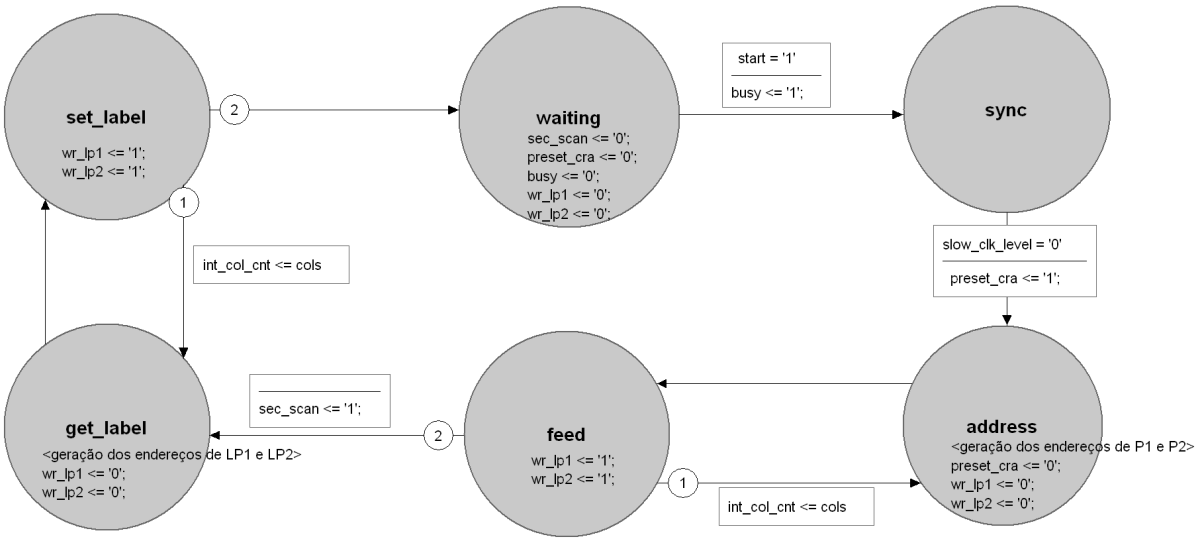


Figura 5.27 – Máquina de estados que constitui o sub-módulo FPLA_CP.

Uma alteração realizada em relação à arquitetura original diz respeito ao modo de processamento da imagem. Originalmente a imagem é processada no modo classificado por este autor como “paisagem”, conforme ilustrado na Figura 5.28(a). Entretanto, a utilização deste modo causa o problema de ordenação dos objetos, previamente discutido na Seção 5.2.2.3, onde a solução apontada foi a utilização de um processo de ordenação a partir da coordenada *x1*.

Entretanto, é possível obter uma ordenação intrínseca dos objetos a partir da utilização do modo de processamento classificado por este autor como “retrato”, conforme ilustrado na Figura 5.28(b), eliminando assim o processo extra de ordenação. O modo de processamento “retrato” consiste no endereçamento das memórias levando em consideração a rotação em 90º no sentido horário da imagem armazenada na memória binária. A mesma ordenação pode ser alcançada a partir do simples armazenamento na imagem rotacionada na memória binária, embora esta estratégia exija a rotação dos objetos em 90º negativos antes dos mesmos serem repassados ao módulo de reconhecimento.

A adoção do modo de processamento “retrato” no módulo FPLA requeriu alterações tanto no sub-módulo FPLA_CP, quanto no sub-módulo FPLA_DP. Dentre as alterações no bloco de controle pode ser destacada a verificação da existência de dados a serem processados a partir do número de colunas da imagem (int_col_cnt <= cols). Já as relativas ao bloco de processamento serão citadas a seguir, na discussão sobre o mesmo.

(a) (b)

Figura 5.28 – Exemplo de processamento em modo (a) paisagem e (b) retrato.

Continuando a discussão sobre o conteúdo dos sub-módulos do módulo FPLA, o segundo e último é o denominado FPLA_DP. Conforme é possível visualizar na Figura 5.29, tal sub-módulo é constituído por três blocos principais, intitulados LBL_ASSIGN, COMB_CIRCUIT e CLASS_REG_ARRAY. De acordo com o algoritmo previamente discutido na Seção 4.2.1.1, numa primeira etapa o bloco LBL_ASSIGN processa dois *pixels* da memória binária por vez (*p1*, *p2*), atribuindo-lhes uma etiqueta inicial, bem como gerando seus respectivos pares equivalentes. Neste sentido, os registradores auxiliares conectados ao bloco implementam a janela de 3x4 *pixels* (Figura 4.6) utilizada no processamento, onde *N* é o número de colunas da imagem. Como apenas um par equivalente ({*pair1_la*, *pair1_lb*} e {*pair2_lx*, *pair2_ly*}) pode ser armazenado por vez na matriz de classes (CLASS_REG_ARRAY), o bloco COMB_CIRCUIT é então utilizado para executar esta tarefa. Neste sentido, a freqüência de operação do bloco LBL_ASSIGN é igual à metade da empregada nos outros dois blocos.

Na segunda etapa de processamento (sec_scan = ‘1’), igualmente dois por vez (*lp1*, *lp2*), as etiquetas previamente armazenadas na memória simbólica são comparadas com o conteúdo do elemento apontado pelas mesmas na matriz de classes. Quando os valores divergem, a memória simbólica é atualizada com aquele contido na matriz de classes.

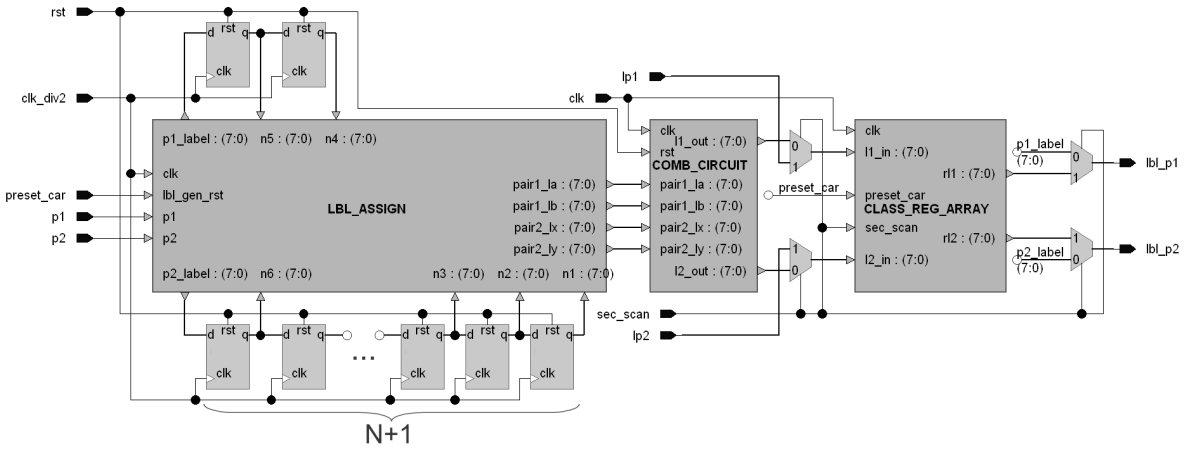


Figura 5.29 – Diagrama de blocos referente ao conteúdo do sub-módulo FPLA_DP.

Descendo um nível agora na hierarquia do sub-módulo FPLA_DP, torna-se possível a visualização dos circuitos que compõem os seus blocos. No caso do bloco LBL_ASSIGN, ilustrado na Figura 5.30, o mesmo é composto por duas unidades de processamento, as quais

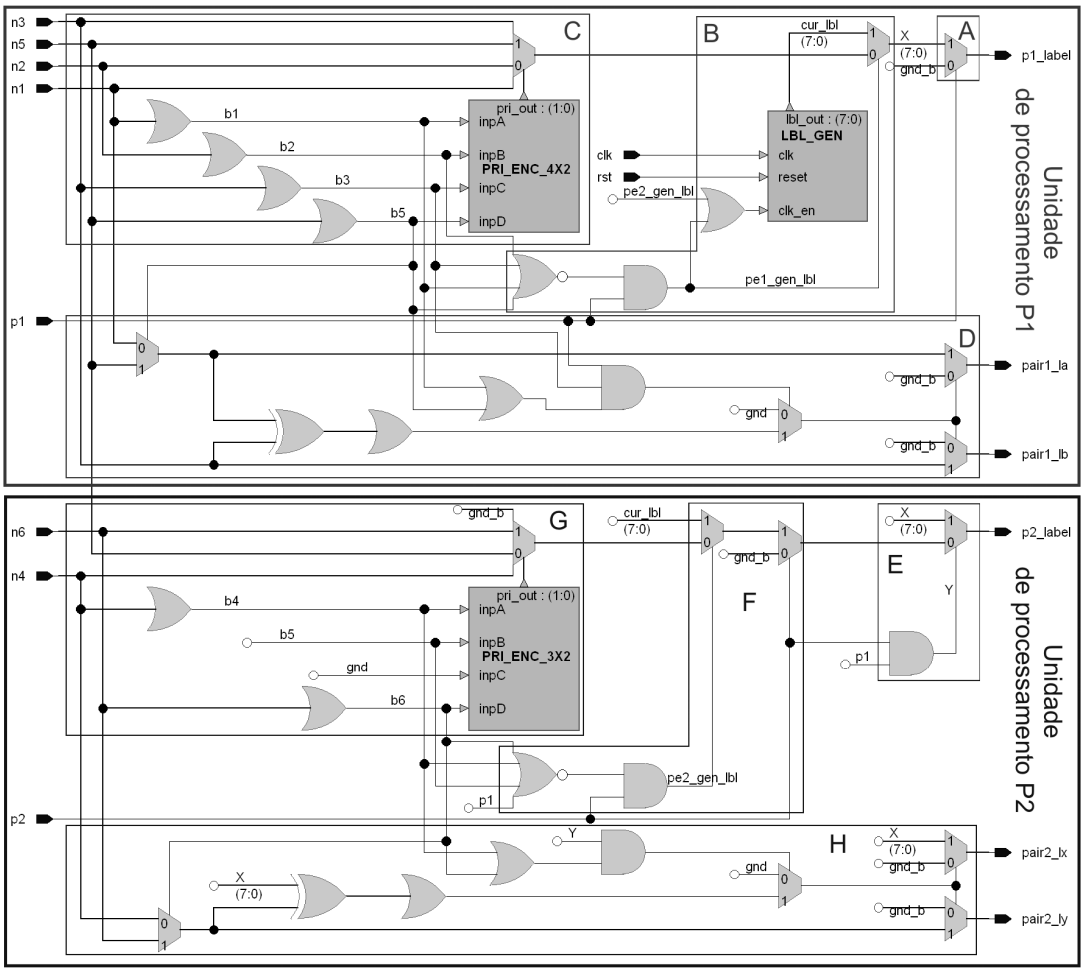


Figura 5.30 – Diagrama esquemático do bloco LBL_ASSIGN.

são responsáveis pela geração das etiquetas e dos pares equivalentes dos *pixels* P1 e P2, respectivamente. No caso da unidade de processamento P1, o multiplexador contido na região "A" utiliza como seletor de canal o valor do próprio *pixel* P1. Quando P1 pertence ao funda da imagem (valor igual a zero), o valor da sua etiqueta é zero, caso contrário, o valor é determinado pelo sub-circuito contido na região "B". Na região "B" é realizada a apuração da etiqueta em função da vizinhança de P1 na janela de processamento (N3, N5, N2, N1). Caso nenhum dos *pixels* vizinhos à P1 possua uma etiqueta, uma nova é gerada a partir do bloco LBL_GEN, caso contrário, o valor da etiqueta é obtido a partir dos vizinhos. Neste sentido o codificador de prioridade representado pelo bloco PRI_ENC_4X2 determina qual dos vizinhos fornece a etiqueta, a partir dos critérios ilustrados na Tabela 5.3. Finalmente, o sub-circuito contido na região "D" é o responsável pela geração dos pares equivalentes relativos ao *pixel* P1. Neste sentido, um par equivalente é gerado quando o valor de P1 for igual a 1 (um), o *pixel* N3 já possuir uma etiqueta, bem como pelo menos um dos *pixels* N5 e N1 também já possuir uma etiqueta, desde que essa última seja diferente da atribuída a N3.

Tabela 5.3 – Tabela-verdade referente aos codificadores de prioridade utilizados no bloco LBL_ASSIGN.

N3	N5	N2	N1	Saída	pri_out(1)	pri_out(0)
1	X	X	X	N3	1	1
X	1	X	X	N5	1	0
X	X	1	X	N2	0	1
X	X	X	1	N1	0	0

Basicamente, a mesma lógica de funcionamento da unidade de processamento P1 se aplica à unidade de processamento P2. A maior diferença reside no fato das novas etiquetas atribuídas à P2 serem geradas também pelo bloco pertencente à unidade de processamento P1 (LBL_GEN). Originalmente, cada unidade de processamento tinha o seu bloco de geração de etiquetas, sendo que a primeira unidade de processamento gerava apenas etiquetas ímpares, enquanto a segunda, apenas pares, no intuito de evitar a sobreposição.

Entretanto, tal solução gerava contadores cujos valores eram pelo menos duas vezes superiores a número real de componentes, haja vista que os mesmos eram incrementados em duas unidades cada vez. Como o valor de uma etiqueta se refere ao índice de um elemento na matriz de classes, quanto maior o valor da etiqueta, maior o tamanho da matriz de classes utilizada no processamento.

Assim sendo, a utilização de um único bloco de geração de etiquetas para ambas as unidades de processamento foi motivada tanto pela economia de recursos, como para facilitar a implementação do processamento da imagem no modo "retrato". Não há empecilhos na

utilização de um único bloco de geração de etiquetas, pois duas etiquetas nunca são geradas ao mesmo tempo para P1 e P2, haja vista que quando P1 e P2 são iguais a 1 (um), a mesma etiqueta atribuída à P1 é utilizada por P2. Originalmente, neste caso, a etiqueta gerada para P2 era simplesmente descartada.

Conforme mencionado anteriormente, até dois pares equivalentes podem ser gerados simultaneamente pelo bloco LBL_ASSIGN, ao passo que o bloco CLASS_REG_ARRAY é capaz de processar apenas um por vez. Neste sentido, o bloco COMB_CIRCUIT é utilizado como uma interface entre os primeiros, repassando apenas um par no momento da geração dos mesmos e postergando o envio do outro para o ciclo seguinte.

Quando dois pares equivalentes chegam ao circuito, apenas o primeiro (*pair1_la*, *pair1_lb*) avança diretamente até os registradores na região “R2”, enquanto o segundo (*pair2_lx*, *pair2_ly*) é armazenado temporariamente nos registradores na região “R1”. No ciclo seguinte, o par armazenado na região “R1” é que avança até a região “R2”, haja vista que o registrador contigo na região “R3” é do tipo J-K, o qual inverte a saída quando ambas entradas são iguais a 1 (um). A mesma ordem de envio dos pares é obedecida caso haja apenas um par equivalente.

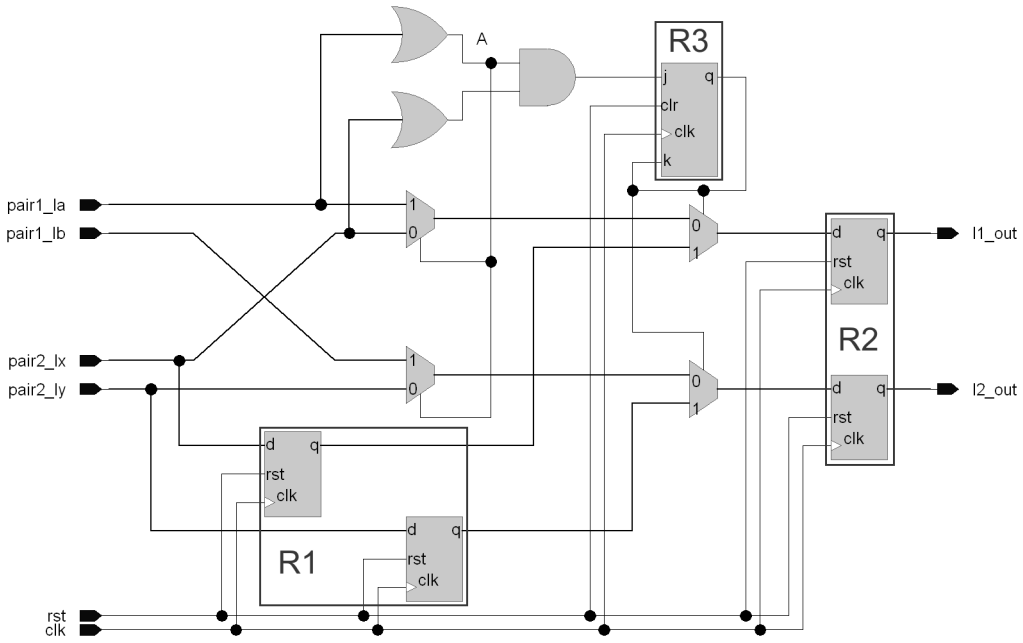


Figura 5.31 – Diagrama esquemático do bloco COMB_CIRCUIT.

O destino final dos pares equivalentes gerados pelo bloco LBL_ASSIGN e reordenados pelo COMB_CIRCUIT é o bloco CLASS_REG_ARRAY, ilustrado na Figura 5.32. Tal bloco é composto por uma matriz unidimensional de elementos, onde *N* é o número

total de elementos na mesma, bem como número máximo de etiquetas suportadas pela arquitetura.

No momento em que o processamento do módulo FPLA é iniciado (preset_car = ‘1’), o valor na porta de saída do registrador contido em cada elemento é ajustado para ser igual ao índice do mesmo na matriz, onde o menor índice é igual a 1 (um). A saída de cada registrador está conectada a dois barramentos (lbl_out_1 e lbl_out_2), que por sua vez, constituem as entradas dos multiplexadores “Mux1” e “Mux2”, respectivamente, cujos seletores de canais são as etiquetas que compõem um par equivalente.

Durante a primeira etapa, conforme os pares equivalentes são recebidos pelo circuito, os valores contidos nos respectivos elementos da matriz são selecionados e classificados de acordo com sua magnitude pelo bloco “Min_Max”. O maior dos dois valores é utilizado como entrada das portas *XOR* contidas nos elementos da matriz, enquanto o menor deles alimenta os multiplexadores existentes nos mesmos elementos. Assim sendo, quando o maior dos valores for diferente do contido atualmente no registrador do elemento, o valor da menor etiqueta é carregado para o mesmo, caso contrário, o valor atual permanece.

A partir deste processo é possível determinar as equivalências entre as etiquetas durante a execução da primeira etapa e utilizar tais equivalências na atualização do conteúdo da memória simbólica na segunda etapa.

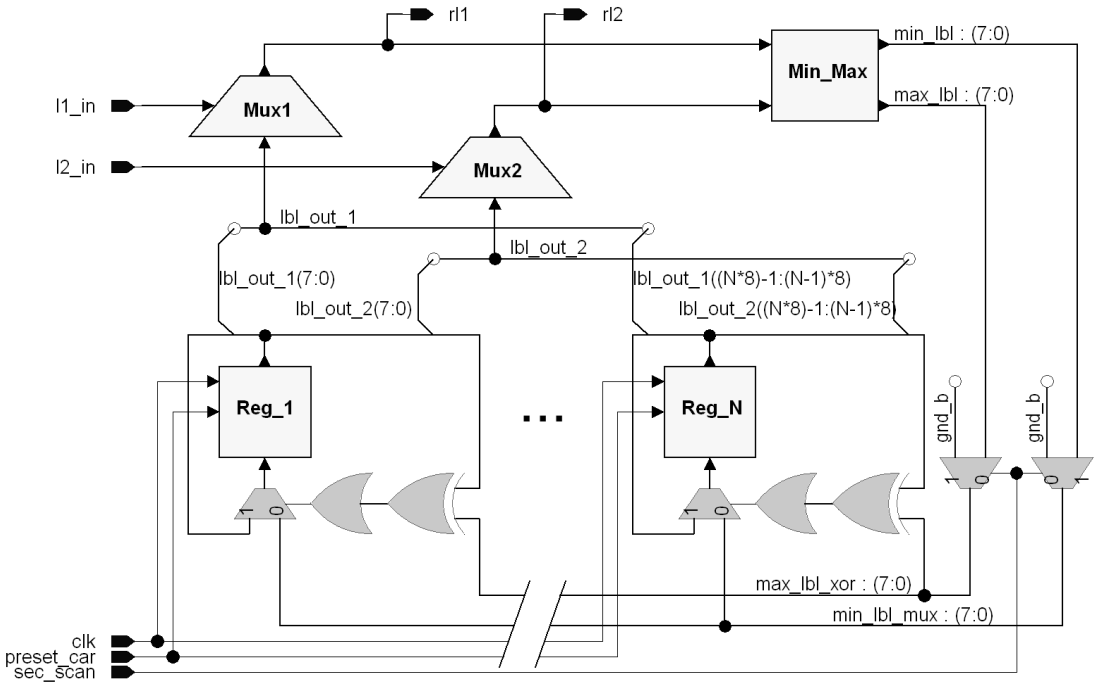


Figura 5.32 – Diagrama esquemático do bloco CLASS_REG_ARRAY.

5.3.2.4 Módulo OCE

Ao término da implementação e validação do módulo FPLA, o próximo construído foi o denominado OCE. Tal módulo, ilustrado na Figura 5.33, opera de forma simultânea à segunda etapa do módulo FPLA, tendo como responsabilidade a extração das coordenadas de cada objeto encontrado pelo segundo módulo mencionado. Basicamente, a tarefa consiste em encontrar as coordenadas esquerda superior e direita inferior do retângulo imaginário que delimita cada um dos objetos. Neste sentido, praticamente todos os dados utilizados são fornecidos pelo módulo FPLA, como o valor simbólico dos *pixels* (*p1_val*, *p2_val*), a sinalização da etapa (*sec_scan*) e as coordenadas de tais *pixels* (*p1_row*, *p2_row*, *p1_col*, *p2_col*).

No que diz respeito à forma como a tarefa é executada, dois valores simbólicos, bem como as coordenadas dos mesmos, são fornecidos por vez. Caso pelo menos um dos valores simbólicos consista numa etiqueta, a mesma é utilizada pelo decodificador na região “A” para a geração de uma saída no estilo *one-hot*, a qual é empregada na seleção do respectivo bloco na região “F”. Como nesta etapa o processamento é realizado com os *pixels* alinhados verticalmente, não há empecilho em determinar a entrada do decodificador a partir de uma operação *OR* entre os dois valores simbólicos. O conteúdo do bloco selecionado na região “F” é então comparado na região “E” com as coordenadas mínimas e máximas provenientes da região “B”. Caso alguma das coordenadas *(x1,y1)* originárias da região “B” for menor do que as atualmente armazenadas, então o conteúdo do bloco na região “F” é alterado, caso contrário, permanece o mesmo. O mesmo se aplica às coordenadas *(x2,y2)*, guardadas as devidas diferenças.

O valor das coordenadas provenientes da região “B” é apurado primeiramente em função dos valores simbólicos de P1 e P2. Caso os mesmos sejam diferentes, o que é apurado pela porta *XOR* na região “A”, os valores fornecidos pela região “B” são provenientes da região “C”, caso contrário, dos comparadores existentes na região “D”. Quando tais valores são provenientes da região “C”, significa que existe apenas uma etiqueta entre os valores simbólicos (o outro é igual a zero), o que implica dos valores mínimos e máximos serem os mesmos. Neste sentido, a região “C” é utilizada para selecionar as coordenadas referentes ao valor simbólico que se constitui numa etiqueta. Já no caso dos valores simbólicos serem iguais, independentemente de maiores que zero (o que é determinado de fato pelo sinal *p_neqz*), as saídas da região “B” são definidas a partir dos comparadores na região “D”.

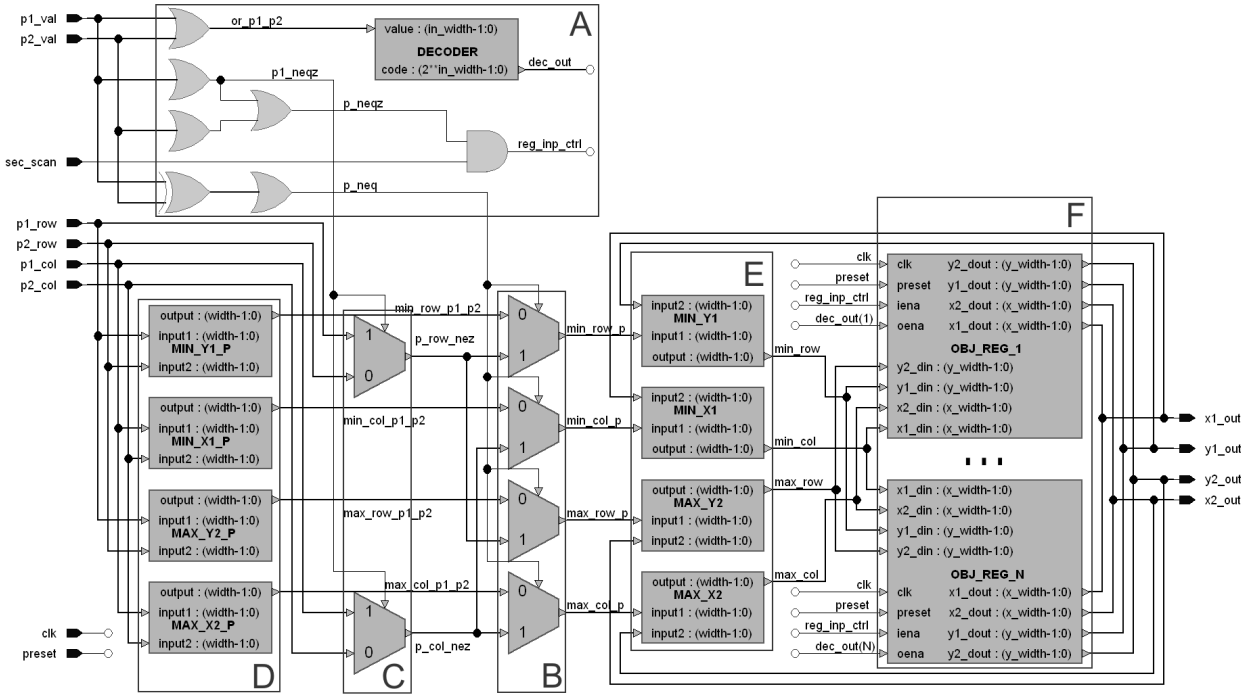


Figura 5.33 – Diagrama de blocos do módulo OCE.

O conteúdo da região “F” consiste numa matriz unidimensional de blocos, onde *N* é o número de elementos que compõem a mesma, de maneira similar à que ocorre no bloco CLASS_REG_ARRAY. De fato, ambas as matrizes devem possuir o mesmo número de elementos, haja vista que cada objeto existente na matriz de classes deve ter uma correspondência na matriz de coordenadas. No que diz respeito ao conteúdo de cada bloco, o mesmo é composto por quatro registradores que armazenam as coordenadas de um objeto, seus respectivos multiplexadores de entrada e *buffers tristate* de saída, conforme ilustrado na Figura 5.34.

Inicialmente (preset = ‘1’) os registradores REG_X1 e REG_Y1 têm seu valor de saída ajustado para o maior valor representável nos barramentos *x1_din* e *y1_din*, respectivamente. Já os registradores REG_X2 e REG_Y2 têm seu valor de saída ajustado para zero. Este artifício é utilizado posteriormente na determinação dos elementos que de fato contém coordenadas de um objeto, haja vista que as atribuídas inicialmente são inválidas.

Conforme as etiquetas são informadas ao módulo OCE, as saídas do respectivo elemento são conectadas aos barramentos de saída (*x1_dout*, *x2_dout*, *y1_dout*, *y2_dout*), a partir da habilitação dos *buffers tristate* pelo sinal gerado pelo decodificador (*o_ena*). Da mesma maneira que ocorre no bloco CLASS_ARRAY_REG, o valor da etiqueta corresponde ao índice do elemento na matriz. Tais saídas são utilizadas de forma externa ao bloco (região “E”) na geração das próprias entradas do mesmo (*x1_din*, *x2_din*, *y1_din*, *y2_din*). Já a

atualização do conteúdo dos registradores a partir destas entradas ocorre quando o bloco está selecionado (o_ena = ‘1’) e pelo menos uma etiqueta foi informada ao módulo OCE (i_ena = ‘1’), considerando que o mesmo esteja ativo (sec_scan = ‘1’).

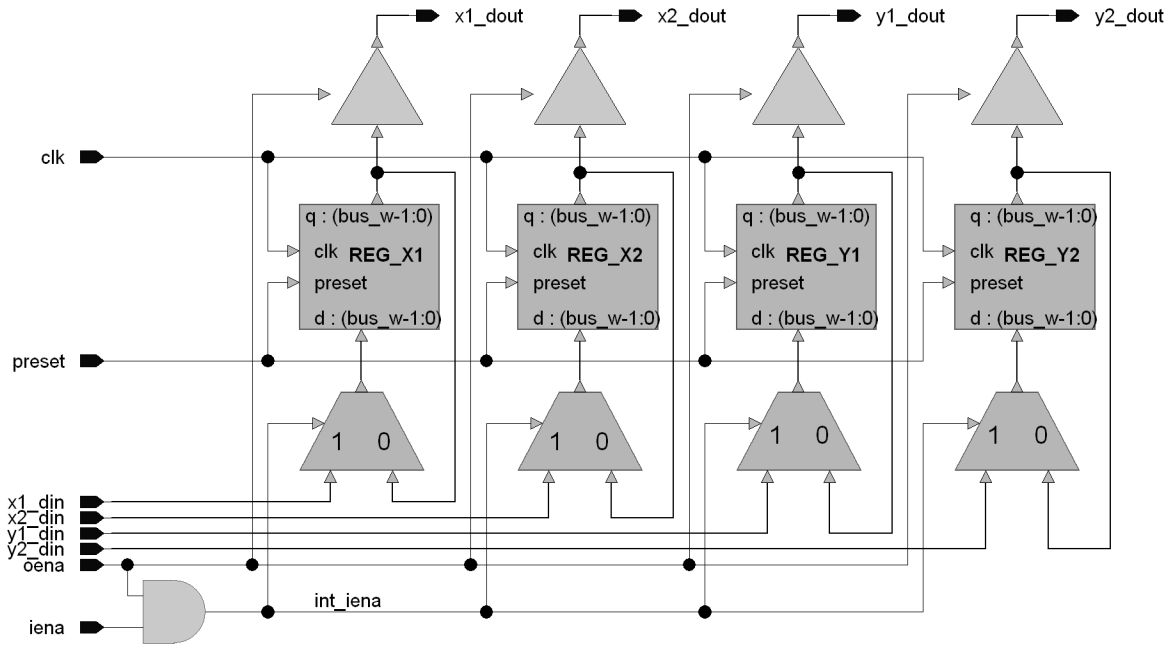


Figura 5.34 – Diagrama esquemático de um elemento da matriz de coordenadas dos objetos.

A utilização de *buffers tristate* foi realizada principalmente no intuito de avaliar o impacto em relação aos resultados da síntese deste bloco e do bloco CLASS_REG_ARRAY, onde se utilizaram multiplexadores na seleção dos canais de saída. Neste sentido, o maior impacto causado pela utilização de *buffers tristate* se refere à liberação de área programável no dispositivo, haja vista que tais estruturas são fisicamente implementadas, tal qual os *cores PowerPC*. No que diz respeito a outros aspectos, como por exemplo a freqüência de operação do circuito, nenhuma alteração significativa foi notada.

5.3.2.5 Módulo ON

Finalizada a implementação do bloco OCE, partiu-se então para a construção do último módulo principal do bloco de *hardware*, denominado ON, cuja tarefa consiste na normalização dos objetos previamente localizados pelo módulo FPLA. Tal módulo é composto pelos sub-módulos ON_CP e ON_DP, bem como pelos circuitos secundários ABS_ADDR_GEN e LBL_FILTER, conforme ilustrado na Figura 5.35.

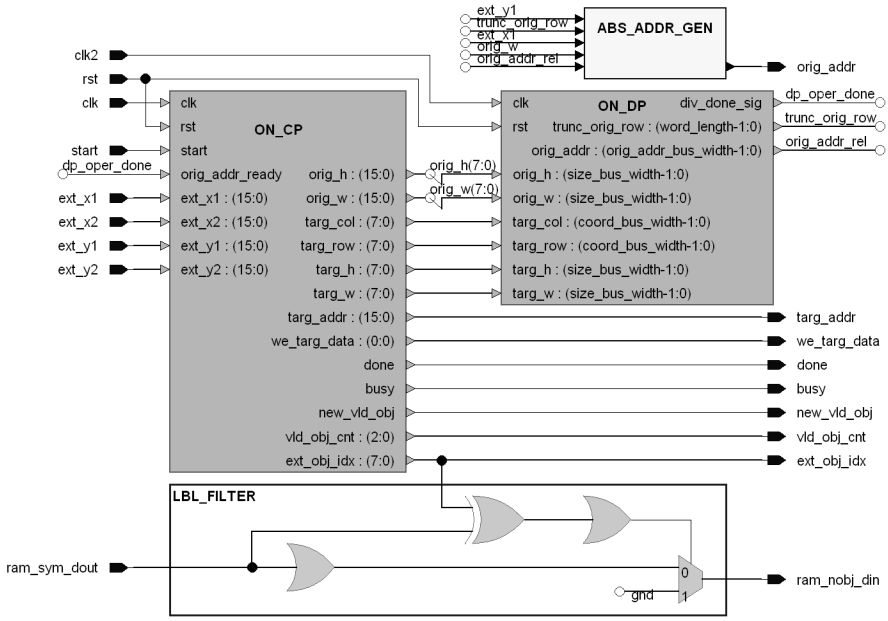


Figura 5.35 – Diagrama de blocos do módulo ON.

O processo de normalização consiste no mapeamento para uma matriz de tamanho fixo dos objetos classificados como caracteres, dentre aqueles encontrados pelo módulo FPLA. Neste sentido, o sub-módulo ON_CP, ilustrado na Figura 5.36, varre a matriz de

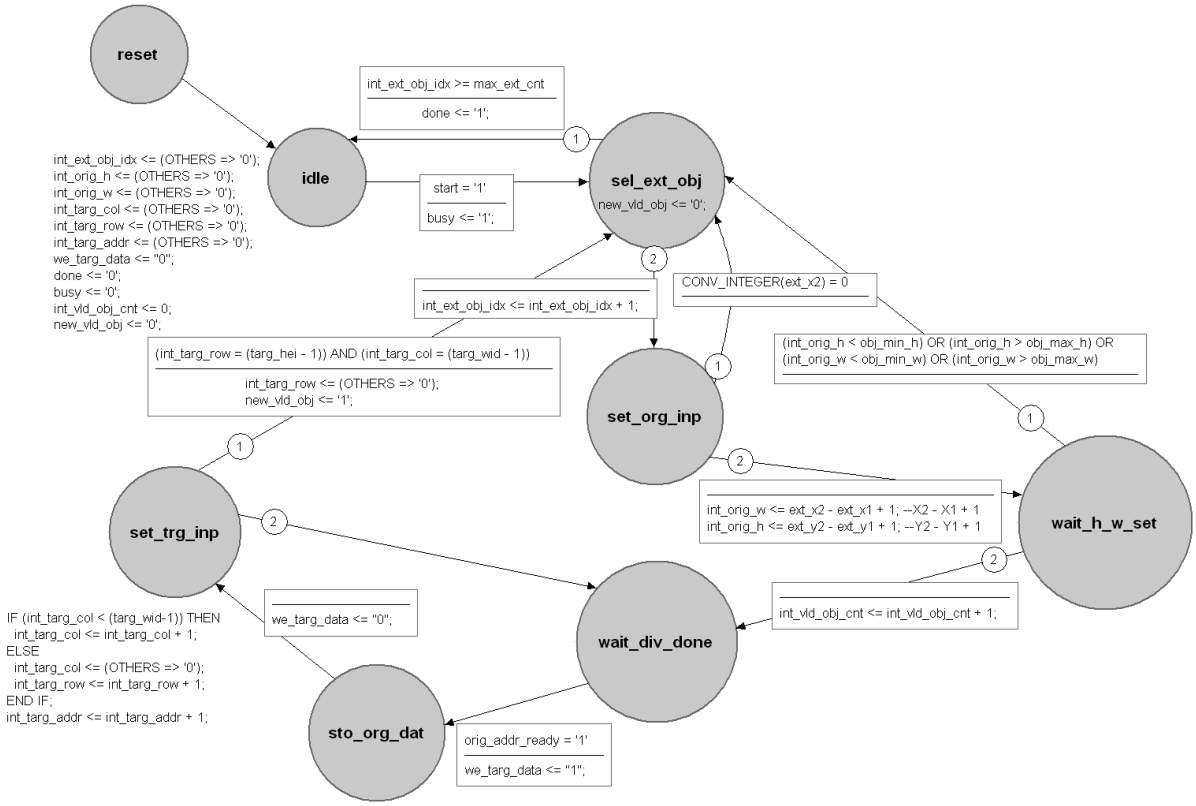


Figura 5.36 – Máquina de estados que constitui o sub-módulo ON_CP.

objetos (módulo OCE) de maneira seqüencial, selecionando as coordenadas cada objeto a partir da especificação do índice do mesmo via barramento *ext_obj_idx*. Tais coordenadas são disponibilizadas ao sub-módulo por intermédio das portas *ext_x1*, *ext_x2*, *ext_y1* e *ext_y2*, sendo então verificadas em relação à sua validade (vide Seção 5.3.2.4) e ao fato de identificarem um caractere ou não (vide Seção 4.2.1). Caso ambas as condições sejam verdadeiras, a altura e a largura do objeto são apuradas a partir das coordenadas do mesmo. Tais medidas, denominadas altura e largura de origem, bem como as mesmas medidas da matriz de normalização, denominadas altura e largura de destino, são então repassadas ao sub-módulo ON_DP por intermédio das portas *orig_h*, *orig_w*, *targ_h* e *targ_w*, respectivamente. A partir de tais informações, o sub-módulo ON_DP apura a razão entre o objeto original e o de destino, sinalizando o término desta operação inicial pelo sinal *dp_oper_done*.

Deste ponto em diante, ON_CP percorre todas as posições existentes na matriz de destino, informando cada uma delas à ON_DP por intermédio das portas *targ_col* e *targ_row*. ON_DP, por sua vez, apura o endereço do *pixel* na imagem de origem (memória simbólica) referente àquela posição na matriz de destino, levando em consideração o algoritmo de interpolação “vizinho mais próximo”. Como o sub-módulo ON_DP não foi projetado levando em consideração que a imagem de origem possa estar contida em outra imagem, o endereço de origem gerado pelo mesmo acaba sendo relativo ao objeto, não à imagem. Neste sentido, o sub-circuito ABS_ADDR_GEN é utilizado para a geração de endereço absoluto do *pixel* na imagem de origem.

Além das coordenadas de destino, ON_CP também gera outros dados necessários ao armazenamento do *pixel* de origem na memória de destino (não ilustrada aqui), como o endereço (*targ_addr*) e o sinal de gravação (*we_targ_data*). Os demais sinais gerados pelo sub-módulo dizem respeito ao status de operação (*done*, *busy*) e *status* dos caracteres normalizados (*new_vld_obj*, *vld_obj_cnt*), estas últimas utilizadas por um bloco externo de *status* do sistema.

No tocante ao sub-circuito LBL_FILTER, o mesmo é utilizado como filtro de gravação da memória de destino. Caso o índice do objeto que está sendo processado (*ext_obj_idx*, que é idêntico à etiqueta do mesmo) for diferente do valor do *pixel* na memória de origem, significa que aquele *pixel* pertence a outro objeto que se encontra nos limites do primeiro, sem entretanto fazer parte do mesmo. Neste caso, o valor zero é gravado na memória de destino.

Conforme mencionado há pouco, uma das tarefas executadas pelo sub-módulo ON_DP consiste em apurar a razão entre a imagem de origem e a matriz de destino. Tal razão

é utilizada no intuito de possibilitar a localização do *pixel* correspondente a cada coordenada da matriz de destino na imagem de origem. Assim sendo, foi necessário o emprego de um circuito divisor, bem como uma forma de representação numérica que suportasse números racionais positivos.

No que diz respeito ao circuito divisor, inicialmente optou-se por uma implementação disponibilizada por Herveille (2006), na qual é empregado o algoritmo de divisão sem restauração. Tal bloco toma um dividendo de *2N bits* e um divisor de *N bits* e disponibiliza um quociente de *N bits*, um resto (inteiro) de *N bits*, além de sinalizar divisões por zero e *overflows*. O primeiro resultado demora *N* bits para ser disponibilizado, enquanto que os subseqüentes são fornecidos a cada novo ciclo. Posteriormente, optou-se pela implementação disponibilizada por Erokhin (2006), a qual possui as mesmas características da anterior, mas disponibilizando os resultados a cada novo ciclo desde o princípio. Em ambos os casos, apenas números inteiros são aceitos.

Já em relação à forma de representação numérica, optou-se pela solução comumente utilizada para os casos de mapeamento de algoritmos escritos em linguagens de alto nível para uma arquitetura de *hardware*: aritmética de ponto fixo. Neste sentido, decidiu-se pela utilização de uma palavra de 26 *bits* para a representação numérica, sendo o comprimento da fração igual a oito *bits*. Dentre os motivos que justificam tal escolha, pode-se citar os seguintes:

- As dimensões máximas de uma imagem de entrada (região da placa) são de 260 x 80 *pixels* (largura x altura);
- As dimensões máximas para um objeto ser considerado um caractere são de 30 x 34 *pixels* (largura x altura). Tanto esta medida, quanto a citada no item anterior, foram apuradas durante a etapa de processamento manual do segundo conjunto de imagens (vide Seção 5.2.2);
- Um comprimento de fração de oito *bits* possibilita precisão suficiente para a representação da razão, até porque o resultado final acaba sendo truncado;
- Como o circuito divisor empregado suporta apenas a divisão inteira, o dividendo deve ser deslocado *N bits* para a esquerda antes da operação ser efetuada, onde *N* é igual ao comprimento da fração. Desta forma, ignora-se o resto e utiliza-se apenas o quociente;
- Considerando-se oito *bits* para fração e nove *bits* para a parte inteira, são necessários mais oito *bits* para o deslocamento anterior à multiplicação,

totalizando assim, 25 *bits*. Como o circuito multiplicador exige que o número de *bits* do dividendo seja exatamente o dobro dos demais termos, adiciona-se mais um *bit* ao dividendo, totalizando assim, 26 *bits*.

A partir destas considerações acerca do sub-módulo ON_DP, torna-se viável a discussão sobre a forma de operação dos blocos internos ao mesmo, ilustrados na Figura 5.37. Inicialmente, as medidas de largura e altura de origem (*orig_w*, *orig_h*) e destino (*targ_w*, *targ_h*) são transformadas de inteiras para ponto fixo pelos blocos na região "A". A alteração de tais entradas implica no cálculo das razões de largura e altura entre as mesmas pelos blocos na região "B", sendo a sinalização de término desta operação apurada e informada ao sub-módulo ON_CP pelo circuito na região "C". Na seqüência, cada par de coordenadas da matriz de destino é também convertido para a representação de ponto fixo pelos blocos na região "D", sendo em seguida utilizado na apuração do endereço de origem.

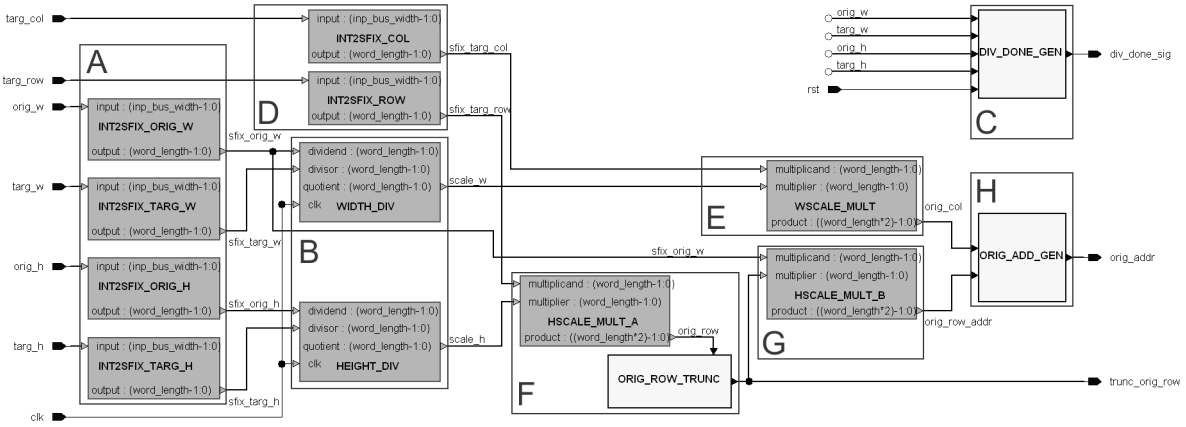


Figura 5.37 – Diagrama de blocos referente ao conteúdo do sub-módulo ON_DP.

As coordenadas referentes à coluna e linha de origem são apuradas nas regiões "E" e "F", respectivamente. A primeira consiste no produto entre a coluna de destino e a razão latitudinal, enquanto última no produto entre a linha de destino e a razão longitudinal. Como o endereço de origem refere-se a uma memória seqüencial, foi necessária uma multiplicação adicional entre a linha de origem (previamente truncada) e a largura de origem, realizada na região "G". Finalmente, o endereço de origem é apurado pela soma das saídas truncadas das regiões "E" e "G".

5.3.2.6 Versão final

Ao término da implementação e validação individual dos módulos FPLA, OCE e ON, os mesmos foram então agregados aos demais, resultando no diagrama de blocos ilustrado na Figura 5.38. Por conta desta junção dos módulos, o conjunto de blocos básicos (Seção 5.3.2.2) sofreu modificações no sentido de inclusão de novos módulos, bem como exclusão e alteração dos já existentes.

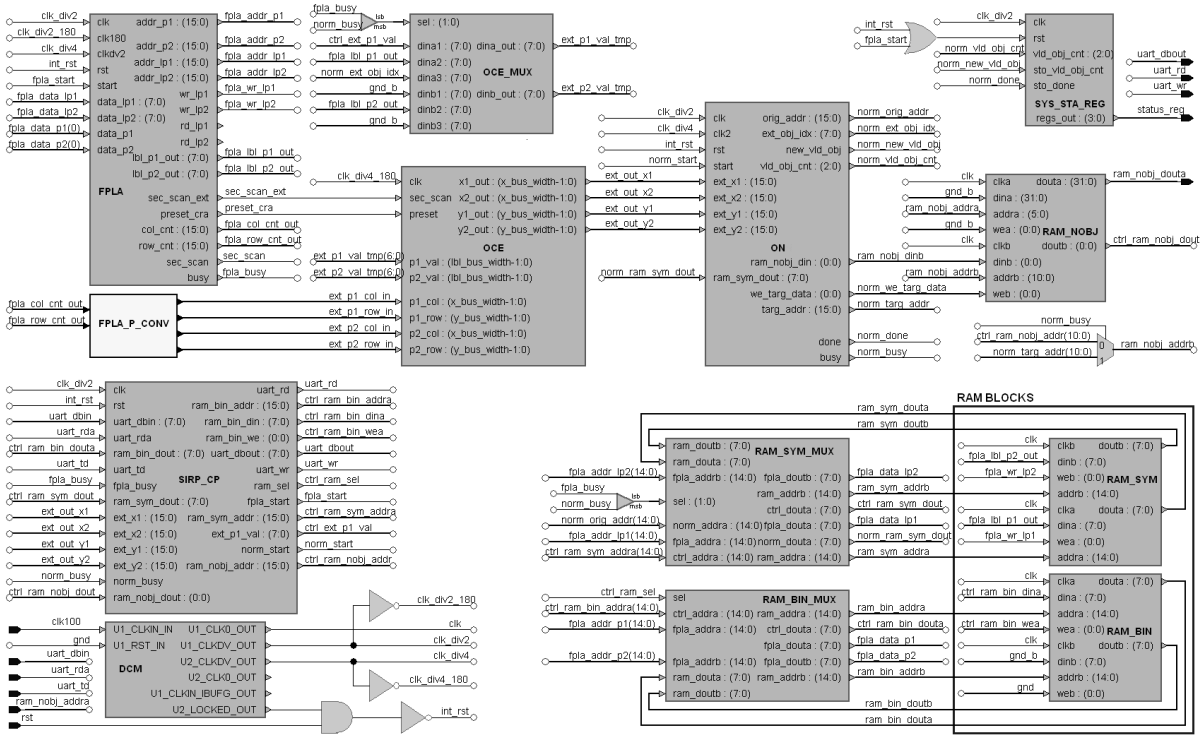


Figura 5.38 – Diagrama de blocos referente ao bloco de *hardware.*

No que diz respeito à exclusão, o módulo UART foi eliminado por conta da maior conveniência de utilização da interface de comunicação serial atrelada à plataforma embarcada básica. Desta forma, além do fato da interface serial tornar-se acessível via *software*, o envio da resposta apurada pelo sistema embarcado (os códigos ASCII da placa) é realizado de uma forma mais simples e ortodoxa. Como o bloco de *hardware* constitui um periférico do sistema embarcado, deixar o gerenciamento do serviço de comunicação a cargo do mesmo seria mais complexo e pouco convencional. Entretanto, na comunicação entre o periférico (bloco de *hardware*) e o sistema (embarcado), o protocolo baseado nos sinais *uart_rda* e *uart_td* foi mantido, haja vista a eliminação do mesmo demandaria um esforço desnecessário para o momento.

Em relação aos módulos alterados, a modificação mais significativa referiu-se ao circuito de geração de *clocks*. O circuito original foi substituído por outro de maior

funcionalidade, denominado *Digital Clock Manager* (DCM), disponibilizado como recurso do próprio FPGA. Dentre outras funções, um DCM é capaz de reduzir da variação da fase de *clock* em diferentes elementos (*clock skew*), bem como derivar diferentes freqüências a partir da multiplicação e divisão de um *clock* principal. Considerando um clock principal de 100 MHz (*clk100*), o módulo DCM foi utilizado na geração de *clocks* secundários com baixa variação de fase de 50 (*clk*), 25 (*clk_div2*) e 12,5 (*clk_div4*) MHz. Versões defasadas em 180º das duas últimas freqüências citadas foram geradas com auxílio de inversores. A redução na freqüência originalmente planejada para os módulos se deveu ao aumento do atraso de propagação causado pela conexão entre os blocos. Outras modificações de menor impacto foram causadas pela inclusão dos sinais referentes aos módulos OCE e ON no módulo de controle, bem como pelo acesso proporcionado ao módulo ON à memória simbólica, via módulo RAM_SYM_MUX.

Finalmente, no tocante às inclusões, três módulos e dois circuitos secundários foram adicionados ao conjunto de blocos básicos. Entre os novos módulos, o denominado OCE_MUX é responsável pela multiplexação dos canais conectados às portas *p1_val* e *p2_val* do módulo OCE. Tais portas são alimentadas pelo módulo FPLA, quando da execução da etapa de etiquetação de componentes conectados, pelo módulo ON, quando da execução da etapa de normalização e pelo módulo SIRC_CP, quando da consulta da matriz de coordenadas. Já o módulo RAM_NOBJ consiste na memória de objetos normalizados, mencionada, mas não ilustrada anteriormente, enquanto o módulo SYS_STA_REG é o responsável por disponibilizar informações de *status* do sistema. A palavra de *status* (*status_reg*) é composta por quatro *bits*, onde os três menos significativos representam o número de caracteres normalizados (zero a sete) e o *bit* mais significativo indica o término do processamento executado pelo bloco de *hardware*.

Já entre os novos circuitos, FPLA_P_CONV é utilizado na conversão das coordenadas geradas por FPLA para as requisitadas por OCE, enquanto o outro circuito seleciona as fontes de endereçamento da memória de objetos normalizados. RAM_NOBJ é endereçada pelo módulo ON durante a execução da etapa de normalização e por SIRP_CP no restante do tempo, permitindo assim a consulta da matriz de objetos normalizados. O primeiro barramento de endereçamento do módulo RAM_NOBJ é exclusivamente conectado à porta *ram_nobj_addr*, habilitando assim o acesso direto à memória para a camada de *software* embarcado.

As principais características arquiteturais do bloco de *hardware* encontram-se resumidas na Tabela 5.4. Dentre as implicações dos valores de tais características, pode-se

citar o tamanho dos módulos de memória. Neste sentido, ambos os módulos RAM_BIN e RAM_SYM são capazes de armazenar até 20.800 palavras de oito *bits*, haja vista que o sistema suporta imagens de entrada com até 260 *pixels* de largura e 80 *pixels* de altura. Embora a memória binária necessite de palavras de apenas um *bit*, optou-se por mantê-las com oito, para o caso de uma futura implementação do processo de etiquetação diretamente sobre imagens em 256 tons de cinza. Já o módulo RAM_NOBJ necessitaria de 1575 palavras de um *bit*, haja vista que até sete objetos de 15 x 15 *pixels* devem poder ser armazenados. Entretanto, o módulo está estruturado em 50 palavras de 32 *bits* cada, no intuito de acelerar o processo de acesso direto à memória mencionado no parágrafo anterior. Apesar de tal estrutura, não há empecilho algum ao uso de palavras de um *bit* na segunda interface de acesso ao módulo (portas *clkb*, *addrb*, *web* e *doutb*).

Tabela 5.4 – Principais características arquiteturais do bloco de *hardware*.

Módulo	Característica	Valor
FPLA	Dimensões máximas da imagem de entrada (largura x altura)	260 x 80
	Número de elementos na matriz de classes	128
OCE	Número de elementos na matriz de coordenadas	128
ON	Comprimento da palavra na representação de ponto fixo (*bits*)	26
	Comprimento da fração na representação de ponto fixo (*bits*)	8
	Dimensões da matriz de saída (largura x altura)	15 x 15

Considerando o FPGA utilizado na implementação do sistema embarcado (um Virtex-II Pro, modelo XC2VP30), um resumo de utilização de recursos do mesmo é apresentado na Tabela 5.5. Tal FPGA possui uma área programável composta por 30.816 elementos lógicos, onde cada elemento lógico equivale a 2,25 *slices*. Cada *slice*, por sua vez, é composta basicamente por uma *Look Up Table* (LUT) de quatro entradas, um *flip-flop* e uma lógica de *carry* associada (XILINX, 2007).

Tabela 5.5 – Resumo de utilização de recursos do FPGA pelo bloco de *hardware*.

	Usados	Disponíveis	Utilização
Recursos lógicos			
Número de *slice flip-flops*	10.532	27.392	38%
Número de LUT's de 4 entradas	9.924	27.392	36%
Número de blocos de RAM de 18K*bits* (Block RAM's)	23	136	16%
Número de multiplicadores de 18x18 *bits* (MULT18x18's)	6	136	4%
Número de *clocks* globais (GCLK's)	5	16	31%
Número de gerenciadores digitais de *clock* (DCM's)	2	8	25%
Distribuição lógica			
Número de *slices* ocupados	9.694	13.696	70%
Número de *slices* contendo apenas lógica relacionada	9.694	9.694	100%
Número de *slices* contendo lógica não relacionada	0	9.694	0%

Já em relação às máximas freqüências de operação do circuito, exibidas na Tabela 5.6, as mesmas foram obtidas após a realização do processo de posicionamento e roteamento no ambiente ISE. Considerando especificamente o FPGA utilizado neste caso, o circuito projetado encontra-se em modo normal de operação. Entretanto, por conta de um dos requisitos de *timming* não ter sido atingido, pode ocorrer que tal condição de operação não possa ser reproduzida em outros dispositivos similares, por conta de variações no processo de fabricação dos mesmos. Neste sentido, uma das formas de sanar tal restrição consiste no emprego da técnica de *pipelining*. A princípio, a utilização de um único estágio já deveria ser suficiente, pois espera-se que a freqüência de operação dobre numa configuração como esta.

Tabela 5.6 – Relatório de restrições de tempo fornecido pela ferramenta ISE.

	Requisitado		Obtido	
Clock	Período(ns)	Freqüência(MHz)	Período(ns)	Freqüência(MHz)
Clk	20	50,0	25,455	39,285
clk_div2	40	25,0	33,480	29,868
clk_div4	80	12,5	62,150	16,090

5.3.3 Bloco de *software*

5.3.3.1 Introdução

Findada a construção do bloco de *hardware*, ainda restava a implementação do bloco de *software* para que o sistema embarcado pudesse ser concluído. Entretanto, a implementação do bloco de *software* estava condicionada à prévia existência da plataforma embarcada sobre a qual o mesmo seria executado. Neste sentido, a elaboração de tal plataforma se deu com o auxílio do assistente *Base System Builder* (BSB), disponibilizado pela ferramenta *Xilinx Platform Studio* (XPS), o qual possibilita a construção de um sistema básico pela especificação de atributos como (XILINX, 2006d):

- tipo de processador (*hard* ou *soft core*, dependendo do tipo de FPGA);
- barramento de comunicação;
- freqüência de operação do processador e do barramento;
- tipo e tamanho de memória volátil;
- configuração de memória *cache*;
- periféricos convencionais (dispositivos de E/S de propósito comum, porta de comunicação serial, temporizadores, etc.);

- fontes de interrupção dos periféricos selecionados.

A plataforma inicial de *hardware* era composta basicamente por dois *hard cores PowerPC*, um módulo RAM de 64 *Kbytes*, um módulo UART a 38.400 *bits* por segundo e um botão de contato momentâneo, utilizado como *reset* externo. Em relação aos processadores, ambos operavam a 100 MHz e não empregavam memória *cache*, sendo que um deles era utilizado apenas para possibilitar a depuração do *software* sobre o outro. O processador sobre o qual o *software* seria executado estava conectado ao módulo RAM por um barramento *Processor Local Bus* (PLB), enquanto o módulo UART e o botão de contato momentâneo conectavam-se ao mesmo via barramento *On-chip Peripheral Bus* (OPB). A freqüência de operação de ambos os barramentos era de 100 Mhz, sendo que a comunicação entre os mesmos era realizada por intermédio de uma ponte PLB pra OPB.

Já em relação à plataforma de *software*, tanto nesta plataforma embarcada inicial, como na versão final da mesma, nenhum SO foi empregado, mas apenas o conjunto de componentes associados à placa de prototipação utilizada (XUP-V2Pro). Tal conjunto, denominado *Board Support Package* (BSP), é composto por itens como *drivers*, bibliotecas, dispositivos de E/S padrão, rotinas de interrupção e outras características de *software* relacionadas (XILINX, 2006c).

A partir desta plataforma foi construído e validado o módulo de reconhecimento de caracteres, conforme discutido a seguir.

5.3.3.2 A RNA sobre plataforma embarcada

Embora a implementação do bloco de *software* pudesse ter sido efetuada a partir da própria ferramenta XPS, optou-se pela utilização de outra mais apropriada para este fim, denominada *Software Development Kit* (SDK). Tal ferramenta, também integrante do ambiente EDK, agiliza o processo de desenvolvimento de *software* embarcado por conta de características como:

- Conjunto avançado de funcionalidades de edição e compilação de código C/C++;
- Gerenciamento de projetos;
- Configuração de *build* e geração automática do *Makefile*;
- Perfeita integração com o ambiente EDK para a depuração e perfilamento de aplicações embarcadas.

Durante o processo de mapeamento da RNA sobre plataforma x86 para a plataforma embarcada, três foram as principais modificações realizadas na mesma. A primeira modificação consistiu numa nova implementação da RNA, a qual veio a substituir também a utilizada sobre plataforma x86. Neste sentido, o intuito foi o de utilizar o suporte à linguagem C++ fornecido pela ferramenta SDK para projetar uma RNA mais compacta e flexível que a original escrita em C. Tal implementação foi trivial, pois contemplava apenas a rede em si, haja vista que os coeficientes haviam sido obtidos pelo MATLAB.

Já a segunda modificação, que consistiu na inclusão dos coeficientes relativos às sinapses das redes no próprio código fonte do programa que as executa, foi realizada por necessidade. Originalmente, tais coeficientes eram armazenados em arquivos de texto, conforme mencionado na Seção 5.2.3, sendo estes arquivos lidos e os coeficientes atribuídos às respectivas sinapses logo após a criação das redes. Entretanto, como no caso do sistema embarcado nenhum SO foi utilizado, não existe a possibilidade de acesso à um sistema de arquivos. Deste modo, a solução foi a criação de constantes matriciais para o armazenamento dos coeficientes, as quais passaram a ser utilizadas no construtor das redes para a atribuição dos pesos às sinapses das mesmas.

Neste ponto a plataforma base sofreu a primeira alteração, pela substituição do módulo RAM interno (utilizava recursos do FPGA) de 64 *Kbytes*, por outro externo, do tipo *Double Data Rate* (DDR) *Synchronous Dynamic Random Access Memory* (SDRAM), de 256 *Mbytes*. Considerando as topologias 225x40x40x26 e 225x20x20x10 para RNAs de letras e números, respectivamente, bem como o fato de cada coeficiente representado em ponto flutuante consumir quatro *bytes*, são necessários 66.960 *bytes* apenas para a seção de dados referente aos coeficientes. Como neste caso o tamanho máximo permitido para a memória de dados é de 64 *Kbytes*, foi necessária a utilização de uma memória externa ao FPGA. Considerando a hipótese de uso do *soft core Microblaze*, a situação é ainda mais desfavorável, haja vista que o tamanho máximo para memória de dados e instruções é de 64 *Kbytes*.

Tão logo tais alterações foram efetuadas, realizou-se uma avaliação do processo de reconhecimento sobre plataforma embarcada, cujo intuito foi o de apurar o tempo de processamento demandado pelo mesmo. Para tanto, foi efetuada a contagem do número de ciclos necessários ao reconhecimento dos sete caracteres de uma placa a partir da utilização de um temporizador conectado ao barramento PLB e o seguinte pseudocódigo:

```
Inicia contador
Para índice de 1 até 3 faça
        Execute ForwardPass da rede de letras (vetor de objetos normalizados[índice])
Fim Para
Para índice de 4 até 7 faça
        Execute ForwardPass da rede de números (vetor de objetos normalizados[índice])
Fim Para
Pára contador
```

onde a função *ForwardPass* consiste na propagação da matriz de entradas pelas camadas da RNA.

Conforme ilustrado na Tabela 5.7, o tempo inicialmente demandado pelo processo foi muito além do esperado, principalmente por conta da inexistência de uma unidade de ponto flutuante no *core PowerPC*. Neste sentido, a primeira tentativa no intuito de reduzir o tempo de processamento consistiu na exploração das opções de compilação do programa, onde (XILINX, 2006a):

- *-g* adiciona informações de depuração no código fonte, o que torna a execução do programa mais lenta;
- *-On* define o nível de otimização empregado pelo compilador, onde $n = 0$ implica em nenhuma otimização, enquanto o nível de otimização máximo é obtido com $n = 3$;
- *-msoft-float* causa a emulação em *software* das operações que empregam ponto flutuante (opção padrão);
- *-mppcperflib* implica no uso de bibliotecas avançadas para emulação de inteiros e ponto flutuante em baixo nível.

Como a exploração de tais opções de compilação não surtiu efeito considerável, outra tentativa consistiu em simular uma redução no número das sinapses das RNAs, o que

Tabela 5.7 – Tempo de execução inicialmente demandado pelo processo de reconhecimento de caracteres sobre plataforma embarcada.

Opções de compilação	Topologia de RNA		Nº ciclos	Tempo (s)
	Letras	Números		
-g –O0 -msoft-float	225x40x40x26	225x20x20x10	285350872	2,85350872
-g –O0 -mppcperflib	225x40x40x26	225x20x20x10	285350825	2,85350825
-O3 -mppcperflib	225x40x40x26	225x20x20x10	252524773	2,52524773
-g –O0 -msoft-float	200x40x40x26	200x20x20x10	247352503	2,47352503
-g –O0 -mppcperflib	200x40x40x26	200x20x20x10	247352532	2,47352532
-O3 -mppcperflib	200x40x40x26	200x20x20x10	217768735	2,17768735
-g –O0 -msoft-float	200x20x20x26	200x20x20x10	185542166	1,85542166
-g –O0 -mppcperflib	200x20x20x26	200x20x20x10	185542137	1,85542137
-O3 -mppcperflib	200x20x20x26	200x20x20x10	163591896	1,63591896

implicou numa redução de 42,66% no melhor dos casos. Entretanto, apesar de tal redução, o tempo demandado ainda era muito além do esperado (alguns milésimos de segundos), além do fato da redução das sinapses da RNAs provocar um impacto negativo na taxa de reconhecimento.

Felizmente, conforme demonstrado por Pratt (2006b), dois simples ajustes podem reduzir em até 10 vezes o tempo de execução de aplicativos baseados em FPGAs Virtex-II Pro e processadores *PowerPC*. O primeiro ajuste consiste no aumento da freqüência de operação do processador, enquanto o segundo, na habilitação das memórias *cache* de instruções e dados. No caso das memórias *cache*, operações realizadas a partir das mesmas representam acessos de ciclo único para a CPU, enquanto operações envolvendo a memória externa acabam por consumir um número muito maior de ciclos em cada acesso. Neste sentido, uma nova bateria de avaliações foi realizada, considerando agora também o *soft core MicroBlaze*. Apenas topologias de rede 225x40x40x26 (letras) e 225x20x20x10 (números) foram contempladas neste novo conjunto de avaliações.

Os novos resultados, como esperado, realmente obtiveram um aumento de até 10 vezes no desempenho referente ao tempo de execução, conforme ilustrado na Tabela 5.8. A partir da análise de tais dados, bem como pela comparação dos mesmos com os obtidos

Tabela 5.8 – Tempo de execução demandado pelo processo de reconhecimento de caracteres após otimizações na plataforma embarcada.

Processador		*Cache* (KB)		UPF	BS	Opções de compilação	Nº ciclos[1]	Tempo(s)
Tipo	Freq. (MHz)	Instr.	Dados					
PPC	100	16	16	N	N	-g -O0 -msoft-float	25295654	0,25295654
PPC	100	16	16	N	N	-g -O0 –mppcperflib	25295671	0,25295671
PPC	100	16	16	N	N	-O3 –mppcperflib	22095847	0,22095847
PPC	300	16	16	N	N	-O3 -mppcperflib	7681319	0,07681319
MB	100	64	64	N	N	-xl-mode-executable -g -O0	64756028	0,64756028
MB	100	64	64	N	N	-xl-mode-executable -O3	61451422	0,61451422
MB	100	64	64	S	N	-xl-mode-executable -g -O0	27623032	0,27623032
MB	100	64	64	S	N	-xl-mode-executable -O3	24027063	0,24027063
MB	100	64	64	S	S	-xl-mode-executable -g -O0	17087825	0,17087825
MB	100	64	64	S	S	-xl-mode-executable -O3	13492318	0,13492318

desempenho foi a utilização de memórias *cache*. Em relação aos parâmetros específicos de cada processador, o aumento da freqüência de operação no *PowerPC*, neste caso, implicou

[1] O período de cada ciclo é de 10 ns, haja vista que o temporizador está conectado ao barramento PLB, cuja freqüência de operação é de 100 MHz.

num ganho proporcional (por um fator de três) de tempo. Já no caso do *MicroBlaze*, a utilização da Unidade de Ponto Flutuante (UPF) se sobressaiu ao emprego do *Barrel Shifter* (BS) e às otimizações em nível de compilação.

Apesar do decréscimo considerável no tempo de execução do processo de reconhecimento proporcionado pela exploração das características arquiteturais dos processadores, 76 ms ainda era um tempo em torno de 10 vezes acima do esperado. Deste modo, partiu-se para a terceira e última grande modificação na RNA original, alteração esta que consistiu na criação de uma versão em ponto fixo do algoritmo de propagação das redes. Neste caso, o ganho em velocidade de processamento se dá por conta do uso exclusivo de números inteiros, sendo que o número de *bits* necessários à representação das partes inteira e fracionária varia conforme a aplicação.

Originalmente a conversão de um algoritmo em ponto flutuante para a respectiva versão em ponto fixo tende a ser um processo senão difícil, ao menos trabalhoso. Entretanto, com o advento de ambientes de suporte ao desenvolvimento, essa tarefa é facilitada. Neste sentido, empregou-se um processo de conversão baseado na metodologia discutida em MATLAB, (2006). Nesta metodologia, o usuário especifica o comprimento das palavras das variáveis e deixa a cargo do pacote de ponto fixo a apuração dos pontos binários, mantendo assim a melhor precisão possível na resposta e evitando a ocorrência de *overflows*.

Tal metodologia, entretanto, aplica-se de forma direta somente a operações aritméticas simples, como adição, subtração e multiplicação, não tendo efeito sobre funções trigonométricas como a tangente hiperbólica, utilizada como função de ativação. Desdobrando a tangente hiperbólica para a sua forma mais simples, tem-se a seguinte equação

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \qquad \textbf{(5.1)}$$

a qual se apresenta ainda num formato um tanto quanto complexo para o seu mapeamento para ponto fixo.

Por conta desta dificuldade, resolveu-se investir numa simplificação da mesma. Conforme mencionado na Seção 2.4.2, a função *piecewise-linear* consiste justamente numa aproximação de um amplificador não-linear, entretanto, opera originalmente na faixa de valores [0, 1]. Decidiu-se então estender a faixa de operação da função *piecewise-linear* original para [-1, 1], originando assim uma aproximação da função tangente hiperbólica. No que diz respeito ao desempenho de tal aproximação, o mesmo é perfeitamente aceitável, conforme é discutido no capítulo de resultados. Na Figura 5.39 encontram-se ilustradas as repostas fornecidas pelas versões original e aproximada da tangente hiperbólica.

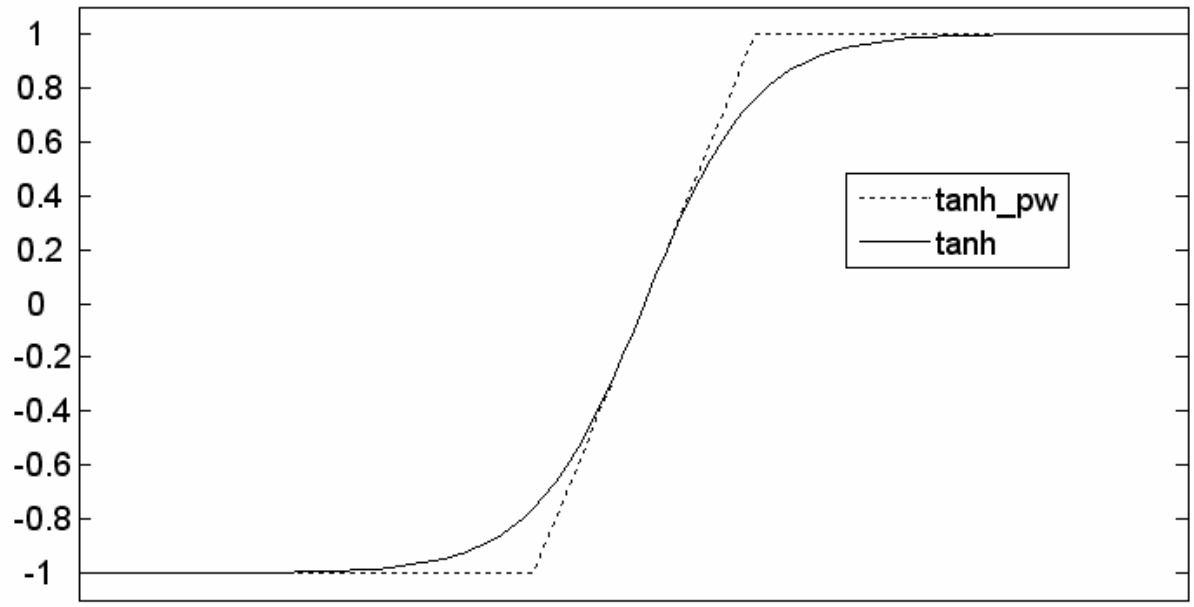


Figura 5.39 – Ilustração das respostas geradas pela tangente hiperbólica original e sua aproximação (*tanh_pw*).

Levando em consideração a versão em ponto fixo do algoritmo de propagação e o emprego de uma aproximação da função de ativação original, foram apurados os resultados exibidos na Tabela 5.9. Considerando tais dados, optou-se pela utilização do processador *PowerPC* e de coeficientes representados em palavras de 16 *bits* na versão final do sistema embarcado, já que o menor tempo foi obtido para este caso. Entretanto, vale à pena ressaltar que no caso da impossibilidade de uso do *hard core PowerPC*, a adoção do *soft core* Microblaze permitiria que o desempenho do sistema se mantivesse muito próximo ao atual.

Tabela 5.9 – Tempo de execução demandado pelo processo de reconhecimento de caracteres após otimizações no algoritmo de propagação das RNAs.

Processador		*Cache* (KB)		UPF	DB	Opções de compilação	Nº ciclos	Tempo(s)
Tipo	Freq. (MHz)	Instr.	Dados					
Comprimento da palavra utilizada na representação dos coeficientes das redes: 32 *bits*								
PPC	300	16	16	N	N	-O3	406573	0,00406573
MB	100	64	64	S	S	-xl-mode-executable -O3	1006711	0,01006711
Comprimento da palavra utilizada na representação dos coeficientes das redes: 16 *bits*								
PPC	300	16	16	N	N	-O3	242443	0,00242443
MB	100	64	64	S	S	-xl-mode-executable -O3	600310	0,00600310

Finalmente, na próxima seção são discutidos os passos restantes na construção do sistema embarcado.

5.3.4 Consolidação do sistema

Tendo ambos os blocos de *hardware* e *software* sido implementados e validados de forma independente, duas tarefas ainda eram necessárias para a conclusão da versão embarcada dos sistema de extração e reconhecimento de caracteres. A primeira delas consistia na agregação do bloco de *hardware* à plataforma embarcada, enquanto a segunda no interfaceamento entre os blocos em si.

No que diz respeito à conexão do bloco de *hardware* à plataforma embarcada, a mesma foi efetuada com auxílio de duas ferramentas distintas: XPS e ISE. Num primeiro momento foi utilizado o assistente de criação e importação de periféricos disponibilizado pela ferramenta XPS para a criação de modelos HDL compatíveis com a interface de periférico utilizada pelo EDK. Durante este processo faz-se necessária a seleção de várias características pertinentes ao periférico, como o tipo de barramento ao qual ele será conectado, os serviços disponibilizados ao mesmo, bem como as configurações de cada um deles.

Dentre os tipos de barramentos, dois são os principais a serem empregados no caso de um processador *PowerPC*:

- OPB: consiste num modelo de barramento projetado para a conexão simplificada de dispositivos de baixo desempenho implementados em *chip*, cujo propósito não é o de conexão direta ao núcleo do processador. O núcleo do processador acessa os periféricos conectados a esse tipo de barramento por meio de uma ponte OPB (IBM, 2007a);
- PLB: consiste num modelo de barramento projetado para conexão direta com núcleo do processador. Geralmente é utilizado na conexão de elementos que necessitam grandes larguras de banda como memórias externas e controladores DMA (IBM, 2007b).

Entretanto, como a interface de tais barramentos é um tanto quanto complexa, o ambiente EDK emprega um modelo que simplifica a conexão dos periféricos aos mesmos.
Tal modelo, ilustrado na Figura 5.40, denomina-se *Intellectual Property Interconnect* (IPIC) e consiste num módulo que permite a conexão entre o núcleo do periférico (User Logic) e o bloco *Intellectual Property Interface* (IPIF), encarregado de conectar o núcleo ao barramento OPB ou PLB. Mais do que a simples conexão, o bloco IPIF disponibiliza serviços requeridos pela maioria dos periféricos, como decodificação de endereços, registradores endereçáveis, gerenciamento de interrupções, etc. (XILINX, 2006b). Desta forma, a conexão do periférico

ao barramento é simplificada, haja vista a complexidade da conexão é resolvida pelo bloco IPIF.

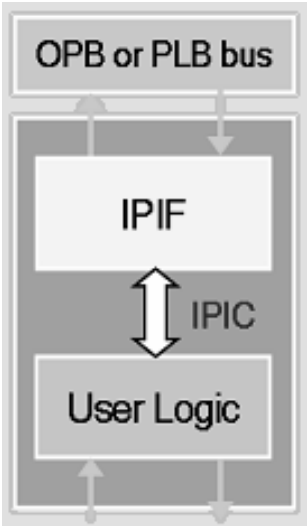


Figura 5.40 – Diagrama ilustrativo da composição e forma de conexão do modelo IPIC.

Na prática, ao final do processo de criação de periférico são gerados dois arquivos HDL, bem como um arquivo de cabeçalho contendo as chamadas a todos os serviços implementados para que os mesmos possam ser acessados por *software*. Em relação aos arquivos HDL um deles se referente ao módulo IPIF, enquanto o outro ao módulo IPIC. Neste sentido, o próximo passo consistiu na instanciação do núcleo do periférico no módulo IPIC, bem como a conexão do mesmo com o módulo IPIF. Essa tarefa foi realizada com o auxílio da ferramenta ISE, sendo que para este caso a comunicação do periférico com o *software* embarcado foi realizada por intermédio de registradores, via barramento OPB. Estando o periférico compatível com o modelo IPIC, a última etapa da conexão do mesmo à plataforma embarcada consistiu na sua importação, por intermédio do mesmo assistente utilizado na sua criação.

Como última tarefa a ser realizada no processo de construção do sistema embarcado, a conexão entre o bloco de *hardware*, agora agregado ao sistema, e o bloco de *software* se deu pela por intermédio da escrita e leitura de registradores. Neste sentido, durante a etapa anterior foram disponibilizados três registrados de 32 *bits* cada, tendo os mesmos sido conectados às portas do bloco de *hardware* conforme ilustrado no Quadro 5.1. No caso das portas cujo comprimento é inferior ao da faixa do registrador disponibilizada para as mesmas, os *bits* sobressalentes (sempre os mais significativos além do comprimento da porta) são ignorados.

Registrador		Porta no bloco de *hardware*	
Nome	Faixa	Nome	Comprimento
Portas de entrada no bloco de *hardware*			
REG0	7:0	uart_dbin	8
	15:8	uart_rda	1
	23:11	uart_td	1
	31:24	ram_nobj_addra	6
Portas de saída no bloco de *hardware*			
REG1	7:0	uart_dbout	8
	15:8	uart_rd	1
	23:11	uart_wr	1
	31:24	status_reg	4
REG2	31:0	ram_nobj_douta	32

Quadro 5.1– Mapeamento realizado entre os registradores acessíveis por *software* e as portas do bloco de *hardware*.

A partir de tal mapeamento dos registradores e do arquivo de cabeçalho disponibilizado na etapa anterior, o *software* responsável pela execução do processo de reconhecimento de caracteres foi modificado e integrado ao bloco de *hardware*. Neste sentido, os bytes recebidos pela interface serial atrelada ao sistema embarcado são repassados ao módulo de *hardware* por intermédio da escrita dos mesmos nas respectivas faixas do registrador REG0. Já as respostas fornecidas pelo bloco de *hardware* para envio pela interface serial são obtidas a partir da leitura das respectivas faixas do registrador REG1. No caso do recebimento de um comando de execução do processo de extração, a camada de *software* identifica o mesmo antes de repassá-lo ao bloco de *hardware* e na seqüência inicia o monitoramento da porta *status_reg* (REG1(31:24)). Quando o término do processo de extração é detectado, os objetos normalizados são lidos da memória de objetos normalizados por intermédio das portas *ram_nobj_addra* (REG0(31:24)) e *ram_nobj_douta* (REG2). Conforme cada objeto normalizado é lido o mesmo é identificado pelo bloco de *software* (RNA) e armazenado num vetor temporário, cujo conteúdo é retornado via serial ao aplicativo de validação sobre plataforma x86 ao término do processo.

No próximo capítulo é apresentada a metodologia empregada na apuração dos resultados de ambas as versões do sistema, bem como uma comparação de desempenho entre as mesmas.

6 RESULTADOS

6.1 Introdução

Findada a implementação de ambas as versões sobre plataforma x86 e embarcada do sistema, partiu-se então para a apuração dos resultados proporcionados pelas mesmas. Tais resultados contemplam tanto a capacidade de identificação das placas de licenciamento veicular, como o tempo demandado para a execução da tarefa.

Entretanto, anterior à apuração de tais dados foi realizada a implementação da opção que permitiu a automatização do processo, haja vista o volume de dados de entrada e saída.

Desta forma, inicialmente é apresentado o método e as considerações envolvidas na tarefa de apuração dos resultados, sendo estes últimos discutidos na seqüência.

6.2 Metodologia de apuração dos resultados

6.2.1 Introdução

Conforme mencionado na Seção 5.2.2, num primeiro momento houve intervenção humana no processamento do conjunto de entradas no intuito de classificar as imagens e disponibilizar dados confiáveis para comparação. Posteriormente, com base nestas classificações e dados de comparação, o conjunto de entrada foi processado de forma automática, provendo os resultados discutidos em seguida.

6.2.2 Os dados de entrada

O conjunto de dados utilizado tanto no fornecimento de parâmetros para a construção da versão final dos sistemas, bem como na obtenção dos resultados fornecidos pelos mesmos é composto por 3.000 entradas. Cada entrada consiste numa imagem da parte traseira de um veículo no momento de sua passagem por um ponto de controle, bem como do respectivo arquivo de coordenadas referentes à possível localização da placa do mesmo na imagem. A Figura 6.1 ilustra algumas das imagens que compõem o conjunto de entradas.

Figura 6.1 – Ilustração de algumas das imagens que compõem o conjunto de dados de entradas.

Entretanto, por motivos que vão desde a existência de elementos que comprometem a visibilidade da placa (desgaste, sujeira, ferrugem, etc.) até a falha em algum ponto do processamento, apenas um subconjunto destas 3.000 entradas pôde ser considerado. Para tanto, durante a etapa de processamento semi-automatizado foram atribuídas classificações a cada uma das entradas, permitindo assim que apenas aquelas onde a placa foi corretamente localizada fossem utilizadas na apuração de resultados.

O método de classificação utilizado é composto por dois níveis, onde o primeiro define classes mais abrangentes e o segundo as mais específicas, conforme ilustrado pelo Quadro 6.1, onde as células contendo o símbolo "–" indicam combinações impossíveis. Neste sentido, dentre os motivos que não conduziram à correta localização de uma placa pode-se citar déficits do algoritmo de localização (Figura 6.2(a)), bem como a inexistência/existência parcial (Figura 6.2(b)) ou comprometimento (Figura 6.2(c)) da placa na imagem. Já em relação aos casos onde a placa foi corretamente localizada, duas situações negativas foram detectadas: visibilidade comprometida ((Figura 6.2(d)-(e)) e déficit do algoritmo de binarização (Figura 6.2(f)-(g)). O descarte sumário de entradas ocorreu nos casos de placas de motocicletas ou com tonalidades invertidas (fundo escuro, letras claras, como na (Figura 6.2(h)).

		Classificação primária			
		Localizada totalmente	**Localizada parcialmente**	**Não localizada**	**Ignorada**
Classificação Secundária	**Totalmente visível**	1112	39	280	–
	Parcialmente visível	–	24	138	–
	Não visível	–	–	140	–
	Comprometida	238	8	492	–
	Binarização deficitária	289	–	–	–
	Motocicleta	–	–	–	147
	Tonalidades invertidas	–	–	–	93
		1639	**71**	**1050**	**240**

Quadro 6.1– Classficações apuradas para o conjunto original de entradas a partir do emprego da classificação de dois níveis.

Assim sendo, do conjunto inicial de 3.000 entradas, 1.391 foram descartadas por conta da impossibilidade de localização completa das placas ou de restrições deste sistema. Das 1.639 entradas restantes, 238 foram desconsideradas por apresentarem comprometimentos que impossibilitariam o seu reconhecimento completo. Desta forma, o conjunto considerado na apuração de resultados é composto por 1.401 entradas. Caso seja levado em consideração que o processo de binarização, embora necessário, não constitui o sistema em si, então o conjunto a ser considerado é composto por 1.112 entradas.

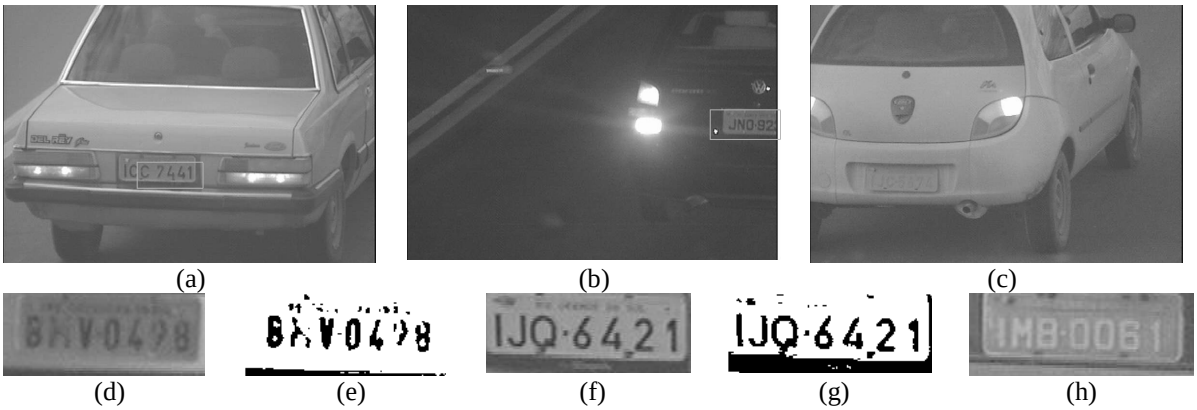


Figura 6.2 – Exemplos de entradas categorizadas a partir do emprego da classificação de dois níveis.

6.2.3 A interface de apuração de resultados

Para que a apuração dos resultados de ambas as versões do sistema pudesse ser realizada de uma forma automática, o primeiro passo consistiu na elaboração de um fluxo comum de processamento. Tal fluxo, conforme ilustrado na Figura 6.3, é composto por uma preparação inicial dos dados de entrada, seguida pelo processamento dos mesmos e culminando com a apuração e armazenamento dos dados de desempenho. No caso da apuração de resultados a partir do sistema embarcado, a etapa de processamento é precedida pelo envio da imagem binarizada para o FPGA via canal RS-232 e sucedida pelo recebimento dos resultados processados no dispositivo pelo mesmo meio. No que diz respeito ao tempo de execução de cada sistema, o mesmo se refere apenas aos blocos responsáveis pelas tarefas de extração de reconhecimento.

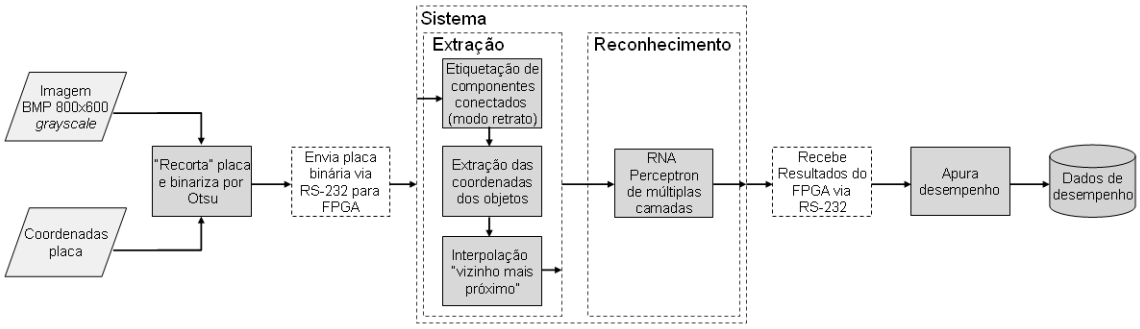


Figura 6.3 – Fluxograma dos processos empregados na apuração dos resultados dos sistemas.

A partir do fluxo discutido acima foi implementada a interface ilustrada na Figura 6.4. Do ponto de vista do sistema sobre plataforma x86, tal opção unifica e automatiza os processos de extração e reconhecimento implementados previamente de maneira separada. Já em relação ao sistema embarcado, tal opção consiste apenas numa forma de fornecimento de dados de entrada e recuperação de resultados.

No que diz respeito aos resultados, os referentes ao desempenho de reconhecimento das placas foram ambos apurados a partir dos padrões de saída fornecidos pelas RNAs. Já em relação ao tempo de execução, a apuração de desempenho é um pouco menos trivial. No caso do sistema embarcado, o mesmo foi obtido sem maiores dificuldades pela utilização de contadores. Entretanto, quando do emprego da mesma abordagem no sistema sobre plataforma x86, corre-se o risco dos resultados serem influenciados por outros aplicativos ou pelo próprio sistema operacional.

Desta forma, no intuito de minimizar ao máximo tal interferência, foi empregado o aplicativo *AQTime*, versão 5.3 (versão de avaliação), fornecido pela empresa *AutomatedQA Corp.* Tal ferramenta consiste num perfilador que possibilita a obtenção de várias medidas de desempenho de um *software*, dentre elas o tempo de execução de cada uma das funções que o constituem. Neste sentido, dentre os vários contadores disponíveis existe um denominado *User Time*, que computa apenas o tempo gasto na execução de código em modo de usuário, ignorando tempo gasto em modo *kernel*, com outras *threads* ou na troca entre as mesmas (AQTIME, 2007). Entretanto, mesmo ignorando o tempo gasto relativo a outras *threads*, a atualização das memórias *cache* por conta de uma troca de *thread* é computada pelo contador *User Time*. Assim sendo, mesmo com a utilização do perfilador, reduziu-se ao máximo o número de aplicações em execução durante a obtenção dos resultados sobre plataforma x86.

Figura 6.4 – Interface da opção construída sobre plataforma x86 para a apuração de desempenho de ambas as versões do sistema.

6.3 Resultados preliminares

A partir do conjunto de entradas discutido na seção anterior, inicialmente foram apurados resultados relativos à capacidade de reconhecimento de caracteres para diferentes configurações de RNA. Neste sentido, foram levados em consideração os algoritmos de aprendizado e as topologias de RNA que apresentaram os melhores resultados ainda na etapa de treinamento (vide Seção 5.2.2.6). Especificamente em relação às topologias de rede, foram avaliadas tanto as de melhor desempenho de reconhecimento, quanto as que exibiram o melhor custo-benefício entre a taxa de reconhecimento e o número de neurônios na RNA. Já no que diz respeito aos resultados em si, os mesmos se referem agora ao reconhecimento completo da placa (sete caracteres), apresentando taxas de acerto para ambas situações onde os erros de binarização são ou não considerados (casos “A” e “B”, respectivamente).

A primeira análise realizada teve como objetivo descobrir a influência dos algoritmos de aprendizagem na capacidade de reconhecimento de placas da RNA. Para tanto, a partir do emprego da metodologia discutida na seção anterior, foram apurados os resultados ilustrados na Tabela 6.1 para o sistema sobre plataforma x86. Analisando tais resultados, nota-se que o algoritmo *traingdx* possibilita uma taxa reconhecimento seis vezes superior à proporcionada por *traincgp* para um caso e uma ligeira vantagem para outro.

No tocante ao primeiro caso, onde a topologia de melhor custo-benefício é empregada, o desempenho consideravelmente inferior do algoritmo *traincgp* se origina na própria etapa de treinamento. Apesar do percentual médio de reconhecimento das RNAs de letras com topologia 225x40x40x26 ser em torno de 60% (78% no caso da rodada escolhida), o desempenho isolado de alguns caracteres comprometeu o processo como um todo, conforme ilustrado na Figura 6.5. Mais do que isso, os casos onde um ou mais caracteres não obtiveram

Tabela 6.1 – Capacidade de reconhecimento de RNAs treinadas a partir dos algoritmos de aprendizagem *traincgp* e *traingdx*, utilizando coeficientes em ponto flutuante.

RNAs			Reconhecimento		
Algoritmo aprendizado	**Topologia**				
	Letras	**Números**	**Nº placas**	**% "A"**	**% "B"**
Traincgp	225x40x40x26	225x20x20x10	104	7,42	9,35
Traingdx	225x40x40x26	225x20x20x10	594	42,40	53,42
Traincgp	225x80x60x20x26	225x60x80x40x10	628	44,83	56,47
Traingdx	225x100x60x40x26	225x80x20x20x10	660	47,11	59,35

um acerto sequer na simulação realizada logo após o treinamento constituem uma regra, não uma exceção, para esta função de aprendizado, comprometendo o uso da mesma.

Já em relação ao segundo caso, o desempenho ligeiramente superior obtido pela função *traingdx* não está obrigatoriamente relacionado ao maior número de neurônios empregados na rede treinada pela mesma. Conforme mencionado anteriormente, neste caso foram consideradas as topologias que possibilitaram o melhor desempenho para cada uma das funções de aprendizado avaliadas. No caso da função *traincgp*, a topologia 225x80x60x20x26 / 225x60x80x40x10 apresentou melhores resultados que outras de ordem superior.

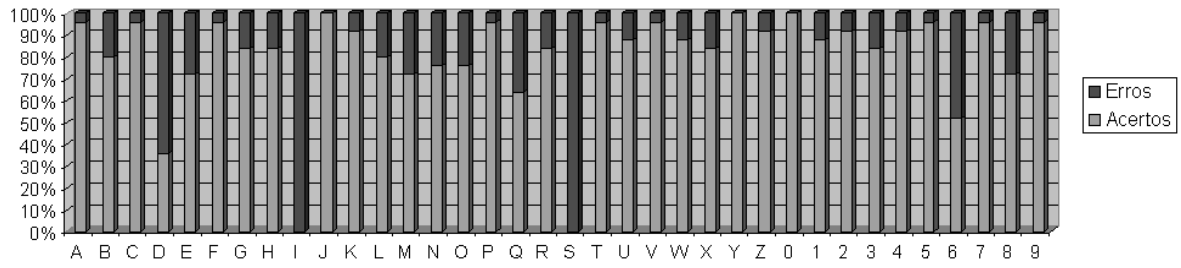


Figura 6.5 – Capacidade de reconhecimento por caractere referente à primeira entrada da Tabela 6.1, apurada a partir de entradas não utilizadas no treinamento, por simulação conduzida logo após a realização do mesmo.

Outro fator analisado foi a influência da utilização da versão aproximada da função de ativação tangente hiperbólica (*tanh_pw*). Neste sentido, manteve-se a mesma configuração das redes utilizada na apuração inicial, inclusive a utilização de aritmética e ponto flutuante, sendo alterada apenas a função de ativação utilizada pelos neurônios. Conforme ilustrado na Tabela 6.2, a influência do emprego da função *tanh_pw* não é significativa, pois reduz em torno de um ponto percentual apenas as taxas obtidas pela utilização da função original.

Na seqüência foi realizada a análise do impacto causado pela utilização de aritmética de ponto fixo na RNA, o que obrigatoriamente implica no uso da função *tanh_pw*. Da mesma maneira que no caso do uso isolado de tal função, o emprego de aritmética de ponto fixo também causa um baixo impacto nos resultados originais (Tabela 6.1), sendo que tal impacto

Tabela 6.2 – Influência do emprego da função tangente hiperbólica aproximada (*tanh_pw*) nos resultados apresentados na Tabela 6.1.

RNAs			Reconhecimento		
Algoritmo de aprendizado	**Topologia**				
	Letras	**Números**	**Nº placas**	**% "A"**	**% "B"**
traincgp	225x40x40x26	225x20x20x10	99	7,07	8,90
traingdx	225x40x40x26	225x20x20x10	578	41,26	51,98
traincgp	225x80x60x20x26	225x60x80x40x10	617	44,04	55,49
traingdx	225x100x60x40x26	225x80x20x20x10	651	46,47	58,54

é da ordem de dois pontos percentuais, aproximadamente. Desta maneira, pode-se concluir que a precisão proporcionada pela utilização de aritmética de ponto fixo e coeficientes de 16 *bits* é muito próxima da obtida pelo uso da representação em ponto flutuante.

Tabela 6.3 – Capacidade de reconhecimento de RNAs treinadas a partir dos algoritmos de aprendizagem *traincgp* e *traingdx*, utilizando coeficientes em ponto fixo de 16 *bits* e a função de ativação *tanh_pw*.

RNAs			Reconhecimento		
Algoritmo de aprendizado	**Topologia**				
	Letras	**Números**	**Nº placas**	**% "A"**	**% "B"**
traincgp	225x40x40x26	225x20x20x10	93	6,64	8,36
traingdx	225x40x40x26	225x20x20x10	570	40,69	51,26
traincgp	225x80x60x20x26	225x60x80x40x10	611	43,61	54,95
traingdx	225x100x60x40x26	225x80x20x20x10	645	46,04	58,00

Conforme demonstrado, as maiores taxas de reconhecimento foram obtidas a partir do emprego de *traingdx* como função de aprendizado, razão pela qual apenas esta função foi considerada nas análises seguintes. Nas avaliações anteriores havia sido considerado apenas o erro quadrático médio (EQM) como critério de parada do treinamento da RNA, bem como o uso de uma camada de entrada composta por 225 neurônios (matriz de 15 x 15). Neste sentido, resolveu-se investigar a possibilidade de aumento da taxa de reconhecimento pelo uso do critério de parada prematura (PP) e o emprego de uma camada de entrada contendo 900 neurônios (matriz 30 x 30).

Levando em consideração o uso de aritmética de ponto fixo, foram apurados os resultados ilustrados na Tabela 6.4. A partir da comparação das duas primeiras entradas desta tabela com as correspondentes na Tabela 6.3, nota-se uma ligeira melhora no caso da topologia de melhor custo-benefício. Já no caso da topologia de melhor desempenho, o ganho se deu tanto pelo aumento da taxa de reconhecimento, quanto pela redução do tamanho da rede. Em contrapartida, quando do aumento da camada de entrada, a utilização do critério de parada prematura gerou um resultado inferior inclusive ao apurado para o caso original (segunda entrada da Tabela 6.3). O mesmo não se aplica para os demais casos onde houve

aumento da camada de entrada, principalmente para as redes de melhor desempenho, que experimentaram ganhos consideráveis. Entretanto, apesar de tal melhora, a hipótese de alteração do tamanho da camada de entrada não será considerada na versão final do sistema. Além do fato de tal alteração implicar na também alteração do bloco de *hardware* (módulo ON e sua respectiva memória), o aumento do tempo de processamento é proibitivo no caso do sistema embarcado, conforme é demonstrado na próxima seção.

Tabela 6.4 – Influência do emprego do critério de parada prematura e do aumento do número de neurônios na camada de entrada na taxa de reconhecimento da RNA, considerando o uso de aritmética de ponto fixo e função de treinamento *traingdx*.

RNAs			Reconhecimento		
Critério de parada	**Topologia**				
	Letras	**Números**	**Nº placas**	**% "A"**	**% "B"**
PP	225x40x40x26	225x20x20x10	574	40,97	51,62
PP	225x60x60x60x26	225x50x50x50x10	629	44,90	56,56
EQM	900x40x40x26	900x20x20x10	593	42,33	53,33
PP	900x40x40x26	900x20x20x10	546	38,97	49,10
EQM	900x40x40x40x26	900x30x30x30x10	670	47,82	60,25
PP	900x50x50x50x26	900x40x40x40x10	708	50,54	63,67

6.4 Resultados finais

Com base nas avaliações discutidas na seção anterior, foram selecionadas três topologias distintas para a apuração dos resultados finais, sendo que em todos os casos a RNA foi treinada com o uso da função *traingdx*, como mencionado previamente. Conforme ilustrado no Quadro 6.2, ambas as versões do sistema foram avaliadas em relação à sua capacidade de reconhecimento de placas de licenciamento veicular e tempo médio de execução desta tarefa.

		RNAs			Reconhecimento		Tempo médio de execução (ms)				
		Critério de parada	**Topologia**		**% "A"**	**% "B"**	**Módulo**				**Total**
			Letras	**Números**			**FPLA**	**OCE**	**ON**	**RNA**	
Plataforma	**x86**	PP	40x40	20x20	40,97	51,62	10,785	0,104	0,008	0,453	11,351
		EQM	100x60x40	80x20x20	46,04	58,00	10,606	0,104	0,008	1,429	12,147
		PP	60x60x60	50x50x50	44,90	56,56	10,904	0,104	0,007	1,056	12,071
	FPGA	PP	40x40	20x20	40,97	51,62	1,675	–	0,412	2,959	5,046
		EQM	100x60x40	80x20x20	46,04	58,00	1,675	–	0,412	9,662	11,748
		PP	60x60x60	50x50x50	44,90	56,56	1,675	–	0,412	7,508	9,595

Quadro 6.2– Resultados finais apurados para ambas as versões dos sistemas, considerando três diferentes topologias de RNA e dois tipos de critérios de parada da etapa de treinamento.

Em relação à capacidade de reconhecimento, nota-se que ambas as versões possuem exatamente o mesmo percentual de acuracidade, comprovando assim a equivalência lógica das mesmas. Entretanto, de uma maneira similar à ilustrada anteriormente pela Figura 6.5, a capacidade de reconhecimento do sistema foi afetada por conta do desempenho isolado de alguns caracteres, conforme ilustrado na Figura 6.6. Neste caso, percebe-se que, proporcionalmente, os caracteres que mais contribuíram negativamente foram o “D” e “O” por parte das letras e “6”, “8” e “9” por parte dos números.

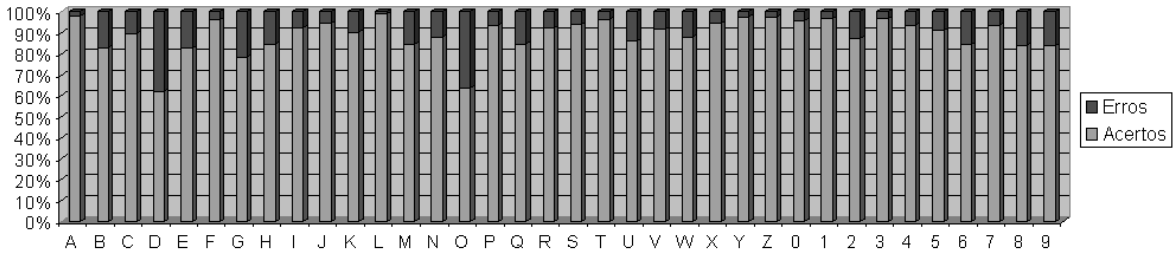


Figura 6.6 – Taxa de erros e acertos por caractere para as RNAs de letras e números de topologias 225x40x40x26 e 225x20x20x10, respectivamente.

A partir de uma análise mais detalhada dos erros e acertos por caractere, verifica-se que, apesar da eliminação dos conflitos entre letras e números pelo uso de RNAs distintas, um número considerável de erros ocorre por conflitos entre os próprios elementos dos grupos distintos. Conforme ilustrado pela Tabela 6.5, no caso do caractere “D”, 36,96% (17 instâncias) do total de erros (46 instâncias) dizem respeito ao conflito de reconhecimento do mesmo com o caractere “D”, apenas. Em contrapartida, houve conflito de reconhecimento do caractere “O” com os caracteres “D” e “Q”em 45%(9 instâncias) do casos de reconhecimento incorreto (20 instâncias).

Os mesmos conflitos apresentados pelo grupo de letras foram exibidos também pelo grupo de números, conforme ilustrado pela Tabela 6.6. Neste caso o percentual de conflito foi ainda maior que o apresentado pelas letras, mas o impacto no resultado final foi menor, haja vista que, proporcionalmente, o número total de erros para os números foi menor que para as letras.

Considerando agora os resultados relativos à velocidade de processamento, para cada entrada do Quadro 6.2 foi apurado o tempo médio de execução de cada um dos módulos que compõem os sistemas. A partir da análise de tais dados, os dois primeiros aspectos percebidos são a considerável redução obtida pelo módulo FPLA na plataforma embarcada e o notável aumento causado por outros dois módulos da mesma plataforma. No que tange o ganho do

Tabela 6.5 – Detalhamento dos erros e acertos por caractere para a RNA de letras de topologia 225x40x40x26.

	D	G	I	M	O
Total instâncias	122	107	1028	262	55
Não reconhecidas	26	16	66	27	11
Conflitos					
A	–	1	1	–	–
B	–	–	–	1	–
C	–	1	–	–	–
D	–	–	2	–	4
G	–	–	–	–	1
H	–	–	–	3	–
I	–	–	–	1	–
K	–	–	–	2	–
L	–	–	1	–	–
N	–	–	–	6	–
O	17	3	–	–	–
Q	2	–	–	–	4
R	1	–	–	–	–
S	–	2	–	–	–
W	–	–	7	1	–
Z	–	–	1	–	–
Total conflitos	20	7	12	14	9
Total erros	46	23	78	41	20
Total acertos	76	84	950	221	35

Tabela 6.6 – Detalhamento dos erros e acertos por caractere para a RNA de números de topologia 225x20x20x10.

	2	5	6	8	9
Total instâncias	393	417	478	423	476
Não reconhecidas	32	18	30	44	49
Conflitos					
0	1	–	3	6	5
1	2	1	–	3	3
2	–	1	1	–	1
3	8	–	–	1	1
4	1	–	8	–	1
5	–	–	6	1	4
6	2	13	–	7	1
7	2	–	–	–	1
8	–	1	26	–	11
9	2	2	–	5	–
Total conflitos	18	18	44	23	28
Total erros	50	36	74	67	77
Total acertos	343	381	404	356	399

módulo FPLA, o mesmo era aguardado, haja vista que justamente no caso da execução repetitiva de tarefas de baixa complexidade aritmética que se esperava obter o maior ganho. Outros fatores que contribuíram para esta expectativa foram o emprego de uma arquitetura tida como estado da arte para esta tarefa e a utilização de uma memória interna ao FPGA, fixando a latência de acesso aos dados de entrada em um ciclo.

Seguindo a seqüência lógica de processamento realizada pelos sistemas, o próximo módulo discutido é o OCE. Por conta da natureza mais enxuta do algoritmo implementado pelo mesmo, foi possível a obtenção de um tempo de execução na ordem de uma centena de microssegundos para a versão sobre plataforma x86. Já no caso da plataforma embarcada, o motivo da inexistência de tempo associado ao mesmo diz respeito à sua execução simultânea com a segunda etapa do módulo FPLA. Entretanto, o tempo de fato consumido pelo mesmo neste caso é de 0,8375 milésimos de segundo, haja vista que cada etapa do módulo FPLA requer metade do tempo total requerido pelo mesmo. Como este tempo esta condicionado pelo bloco FPLA, acredita-se que quando considerado de forma independente o módulo OCE do sistema embarcado possa ser tão rápido quanto ou até mais que a sua versão sobre plataforma x86.

Já no caso do módulo ON, a vantagem da versão sobre plataforma x86 reside no fato da grande demanda aritmética do algoritmo de redimensionamento. Neste caso prevaleceu o poder de processamento disponibilizado pela ULA do processador Pentium IV utilizado na plataforma. Mais do que isso, a vantagem só não foi maior por conta da unidade de divisão com latência de um ciclo empregada na plataforma embarcada.

Finalmente, em relação ao módulo de reconhecimento, apesar de proporcionalmente menor, a mesma desvantagem apresentada pelo módulo ON aplica-se também a este caso. O que contribuiu significativamente para a redução da desvantagem do sistema embarcado nesta questão foi o emprego de aritmética de ponto fixo, conforme demonstrado no capítulo relativo à implementação dos sistemas.

7 CONCLUSÕES

Tendo como a principal motivação a franca expansão dos mercados de sistemas embarcados e visão computacional, o intuito deste trabalho foi o de desenvolver um sistema voltado à extração e reconhecimento de caracteres ópticos, baseado na metodologia de co-projeto de *hardware* e *software*.

Como resultado deste estudo têm-se duas versões funcionais do sistema proposto, sendo uma completamente em *software*, sobre plataforma x86, e a outra composta por uma combinação de blocos de *hardware* e *software*, implementada sobre um FPGA. Além disso, este trabalho proporcionou a publicação de dois artigos, conforme disposto no Anexo A, e a possibilidade de escrita de pelo menos mais um, referente ao sistema como um todo.

No que diz respeito ao processo de desenvolvimento do mesmo, a primeira etapa consistiu na revisão teórica acerca de ambas as áreas envolvidas, bem como na escolha de um sistema para ser utilizado como estudo de caso. Por conta da vasta lista de aplicações de Sistemas de Reconhecimento de Placas de Licença Veicular, que inclui desde o controle de estacionamento até a identificação de veículos roubados, um sistema deste tipo foi selecionado com estudo de caso. Entretanto, por conta da complexidade de um SRPLV, este trabalho considerou a implementação da segunda (extração de caracteres) e terceira (reconhecimento de caracteres) etapas do mesmo, enquanto a primeira (localização da placa) foi alvo de estudo de outro trabalho (Pacheco, 2007).

Para que a versão embarcada do sistema pudesse ser implementada, inicialmente foi necessária a construção de uma versão totalmente em *software*, tarefa esta realizada com o emprego da linguagem C++, sobre uma plataforma x86. Além de agilizar o processo de validação dos algoritmos previamente selecionados para comporem o sistema final, a versão em *software* também foi construída para servir como parâmetro na comparação com a versão embarcada. Outra possibilidade não explorada de comparação seria entre o sistema heterogêneo aqui desenvolvido e uma versão totalmente em *software*, construída sobre a mesma plataforma embarcada. Do ponto de vista arquitetural, ao menos, seria uma comparação mais justa.

Dentre as conclusões obtidas a partir da versão de avaliação do sistema, a mais importante foi a de que a utilização da região candidata no formato em que a mesma é disponibilizada por Pacheco (2007), implica diretamente no aumento da complexidade das etapas posteriores. O fato da região candidata compreender uma área que se estende além dos

limites da placa, em ambas as direções horizontal e vertical, acaba por influenciar negativamente o resultado do sistema como um todo.

O primeiro procedimento a ser afetado é a binarização, que apesar de estar fora do escopo deste trabalho, foi aqui empregado como um processo auxiliar e geralmente está presente em sistemas de visão computacional. Conforme discutido na Seção 5.2.2.2, a existência de formas além dos limites da placa pode causar a conexão indevida das mesmas com os caracteres durante a binarização, o que geralmente implica no não reconhecimento do caractere e, por conseqüência, da placa. Outro fato que evidencia tal problema é o considerável número de entradas do processo de avaliação que foram comprometidas ainda na etapa de binarização, conforme ilustrado no Quadro 6.1.

De acordo com a discussão realizada na Seção 5.2.2.5, problema similar ocorre quando da tentativa do uso do método de projeções na etapa de identificação de regiões. Por conta da existência de objetos nas extremidades da região candidata, a separação dos caracteres fica comprometida , haja vista que a determinação do ponto de início e fim dos mesmos pelo método de picos e vales é impossível em certos casos, como o ilustrado na Figura 5.9.

Entretanto, acredita-se que a solução da dificuldade causada pelo formato da região candidata não resida na alteração do método pelo qual a placa é localizada, mas sim na realização do enquadramento da mesma, após a localização da região inicial. Um exemplo de método empregado para esta finalidade é o apresentado por Dias (2007), discutido na Seção 2.5.7. Tal método realiza inclusive a correção da distorção de perspectiva, o que sugere que o mesmo não seja de implementação trivial. Apesar disso, entende-se que o retorno obtido na construção de um módulo que implemente método igual ou similar supere qualquer outra tentativa de correção nas etapas posteriores, como foi o caso da tentativa do uso do método de força bruta, discutido nas Seções 5.2.2.3 e 5.2.2.6. Já em ambientes controlados, como o da aplicação de identificação de objetos numa linha de produção, a etapa de correção de ângulo da imagem talvez fosse dispensável, o que diminuiria a complexidade do módulo de enquadramento.

No que diz respeito à etapa de reconhecimento, um número muito maior de possibilidades puderam ser avaliadas em função da utilização do ambiente MATLAB. Neste sentido, a escolha dos resultados fornecidos pela função de treinamento *traingdx* foi feita com base numa série de resultados, como os ilustrados da Tabela 6.1 à Tabela 6.3, que sempre a indicaram como possuindo o melhor desempenho para este caso. Outra conclusão obtida a partir da análise das tabelas do Capítulo 6, foi que a relação de custo-benefício proporcionada

pelo aumento de número de neurônios, tanto na camada de entrada, quanto nas camadas escondidas, é consideravelmente baixa para este caso. Também neste caso é insignificante o ganho propiciado pelo emprego do critério de parada prematura, conforme demonstrado pela comparação entre a Tabela 6.3 e a Tabela 6.4.

Já em relação à intenção inicial de utilizar aritmética de ponto flutuante na RNA, a mesma demonstrou-se inviável, haja vista que esta abordagem compromete seriamente o tempo de execução da etapa de propagação no sistema embarcado. Como solução, optou-se pelo uso de aritmética de ponto fixo, bem como de uma aproximação da função de ativação original (tangente hiperbólica). Considerando a utilização de 16 *bits* na representação dos pesos das sinapses, obteve-se uma redução de praticamente 36 vezes no tempo inicialmente necessário à execução da etapa de propagação, quando da utilização de aritmética de ponto flutuante na mesma. No tocante ao impacto na precisão das respostas, o mesmo se demonstrou perfeitamente aceitável, pois houve um decréscimo total médio de apenas dois pontos percentuais nas taxas de reconhecimento originais.

A opção pelo processador *PowerPC* se justifica principalmente pelo fato do mesmo ser capaz de executar a etapa de propagação num tempo praticamente três vezes inferior ao requerido pelo *soft core MicroBlaze*. Além disso, por se tratar de um *hard core*, sua utilização na implica em consumo de área programável do FPGA, cujo percentual de utilização foi de 70%, considerando apenas o bloco de *hardware*. Entretanto, conforme mencionado na Seção 5.3.3.2, pode-se optar pelo *soft core MicroBlaze* sem que isso incorra numa degradação comprometedora no tempo de execução do sistema. Esta opção possibilita o emprego de FPGAs que não possuem *hard cores* implantados no seu interior na implementação do sistema aqui proposto. Além disso,acredita-se que o emprego de versões posteriores da aqui considerada para o *core MicroBlaze* (4.0) sejam capazes de alcançar freqüências de operação mais elevadas, diminuindo ainda mais a diferença existente em relação ao *core PowerPC.*

Finalmente, em relação ao sistema como um todo, têm-se os resultados ilustrados no Quadro 6.2. A partir de tais resultados verifica-se que, no pior caso, o tempo de execução da versão embarcada é ligeiramente inferior ao da versão sobre plataforma x86, enquanto no melhor caso, é 2,25 vezes superior. Neste caso, o principal fator que contribui para o desempenho do sistema embarcado é notavelmente o número de neurônios utilizados nas camadas escondidas da RNA. Este fato já era esperado, haja vista que o processamento da RNA é realizado de uma forma seqüencial, o que favorece o processador sobre plataforma x86, cuja freqüência de operação (3 GHz) é 10 vezes superior à do processador utilizado na

plataforma embarcada. O mesmo ocorreu em relação do módulo de normalização (ON), o qual demanda um elevado índice de processamento aritmético.

Entretanto, a principal fonte de redução no tempo de processamento da versão embarcada consiste nos módulos responsáveis pela etiquetação de componentes conectados (FPLA) e extração de coordenadas dos objetos (OCE). Neste caso, a maior possibilidade de paralelização e a baixa demanda aritmética dos algoritmos favoreceram a versão embarcada, mesmo a freqüência de operação destes blocos não superando 25 MHz. Na verdade, praticamente todo o bloco de *hardware* opera numa freqüência igual ou inferior a 25 MHz, já que apenas os módulos RAM operam a 50 MHz.

Já em relação à capacidade de reconhecimento exibida pelo sistema, esperavam-se obter taxas maiores. Mesmo com a utilização de RNAs distintas para letras e números, ainda persistiram problemas de conflitos de reconhecimento, neste caso entre elementos do mesmo grupo (letras ou números). Entretanto, como esta constitui a primeira versão do sistema, não havia como considerar a utilização de inúmeros outros métodos empregados no intuito de aumentar a taxa de reconhecimento. Preferiu-se optar pela implementação de uma versão funcional, com um percentual de reconhecimento mediano, e investir na avaliação de alternativas que possibilitem a melhor relação entre o incremento da taxa de reconhecimento e o esforço de implementação. Neste sentido, conforme mencionado anteriormente, acredita-se que a utilização de uma etapa de enquadramento da placa, após a localização inicial da mesma, forneça o melhor custo-benefício.

Como última consideração, convém mencionar que, conforme discutido nos três últimos parágrafos, as heurísticas adotadas no particionamento do sistema embarcado, apesar de darem margem a considerações, em geral proporcionaram o resultado esperado. Entretanto, acredita-se fortemente que o particionamento de um sistema como o implementado aqui possa ser realizado de uma forma mais eficiente pela utilização de uma abordagem como a proposta por Davis (2005). Nesta abordagem, inicialmente o sistema é implementado totalmente em *software* sobre a plataforma embarcada, a qual consiste num FPGA. Em seguida, é realizado o perfilamento da aplicação em busca dos gargalos de processamento do sistema. Deste modo, consegue-se uma ótima relação entre aumento de desempenho e esforço de implementação, pois no caso mais extremo, apenas seções de um algoritmo podem vir a ter implementações dedicadas.

Finalizando a discussão e este capítulo, seguem abaixo sugestões de trabalhos futuros que certamente contribuirão com o aprimoramento desta primeira versão do sistema:

- Incorporação do processo de binarização à versão embarcada do sistema. Eventualmente pode-se também empregar técnicas de identificação de regiões que trabalhem diretamente sobre imagens em tons de cinza, o que praticamente descartaria a necessidade de prévia binarização da imagem de entrada;
- Implementação de um processo de enquadramento da região de interesse contida na região candidata inicial. Mesmo que tal processo não seja capaz de realizar a correção de distorção angular da placa, acredita-se que já se obtenha um ganho considerável, haja vista que a distorção angular pode ser corrigida pelo próprio posicionamento da câmera, em algumas situações;
- Avaliação do ganho proporcionado pela utilização de técnicas de processamento no domínio da freqüência para o caso do módulo de normalização. Como este módulo implementa uma operação de interpolação, a mesma pode ser realizada a partir de filtros digitais de duas dimensões. Tal implementação pode vir a proporcionar um ganho, ao menos na velocidade de execução, em comparação ao processamento no domínio do espaço realizado atualmente;
- Avaliação do emprego de métodos de classificação sintáticos. Como os mesmos se baseiam em gramáticas e regras de formação, talvez sejam mais propícios para uma implementação em *hardware*. Imagina-se que no caso das gramáticas não demandarem muita área para o seu armazenamento, a implementação das regras de formação poderia ser feita a partir de máquinas de estado;
- Integração do módulo dedicado de localização de regiões candidatas implementado por Pacheco (2007) ao sistema embarcado aqui desenvolvido. Tal integração, juntamente com uma solução para a questão da binarização, resultam numa versão completa de um SRPLV embarcado.

BIBLIOGRAFIA

ADAMS, J. K.; THOMAS, D. E. **The design of mixed hardware/software systems**, 33rd Design Automation Conference Proceedings, 1996. p. 515-520.

ALOIMONOS, Y.; ROSENFELD, A. Computer Vision. **Science**, EUA, v. 253, p. 1249-1254, 1991.

ANAGNOSTOPOULOS, C. N. E.; ANAGNOSTOPOULOS, I. E.; LOUMOS, V.; KAYAFAS, E. A License Plate-Recognition Algorithm for Intelligent Transportation System Applications, **IEEE Transactions on Intelligent Transportation Systems**, v. 7, p. 377-392, 2006.

ANDREADIS, I. & AMANATIADIS, A. **Digital Image Scaling**, Proceedings of 2005 IEEE Instrumentation and Measurement Technology Conference (IMTC 2005), 2005, v. 3, p. 2028-2032.

AQTIME, **AQTime 5 Help** – seção *Counters Overview*. Documentação da versão de avaliação da ferramenta AQTime 5.3, 2007.

AYLAND, N.; DAVIES, P. **Automatic vehicle identification for heavy vehicle monitoring**, Second International Conference on Road Traffic Monitoring, 1989. p. 152-155.

BARROSO, J.; BULAS-CRUZ, J.; DAGLESS, E. L. **Real-Time Number Plate Reading**, 4th IFAC Workshop on Algorithms and Architectures for Real-time Control, 1997.

BERGER, A. S. **Embedded Systems Design:** An Introduction to Processes, Tools, and Techniques. EUA: CMP Books, 2001. 237 p.

BREMANANTH, R.; CHITRA, A.; SEETHARAMAN, V.; NATHAN, V. S. L. **A robust video based license plate recognition system**, Proceedings of 2005 International Conference on Intelligent Sensing and Information Processing, 2005. p. 175-180.

CAMPOS, T.; BAMPI, S.; SUSIN, A. **Sistema de Identificação de Placas por Processamento Automático de Imagens**, VII Workshop IBERCHIP, 2001.

CHIODO, M.; GIUSTO, P.; JURECSKA, A.; HSIEH, H. C.; VICENTELLI, A. S.-; LAVAGNO, L. Hardware-Software Codesign of Embedded Systems. **IEEE Micro**, v. 14, p. 26-36, 1994.

COELHO, C. J. N., Jr.; SILVA, D. C. D., Jr.; FERNANDES, A. O. **Hardware-software codesign of embedded systems**, Proceedings of XI Brazilian Symposium on Integrated Circuit Design, 1998. p. 2-8.

COETZEE, C.; BOTHA, C.; WEBER, D. **PC Based Number Plate Recognition System**, Proceedings of 1998 IEEE International Conference on Industrial Electronics, 1998.

COWELL, J.R. Syntactic pattern recognizer for vehicle identification numbers. **Image and Vision Computing**, Elsevier Science, v. 13, p. 13-19, 1995.

CRAVOTA, R. **2007 EDN Microprocessor Directory**. Disponível em: http://www.edn.com/index.asp?layout=mpd2007&industryid=48351. Acesso em: 25 nov 2007.

DAVIS, D.; SRINIVAS, B.; RANJESH, J. **Hardware/Software Codesign for Platform FPGAs**, 2005 Embedded Systems Conference, San Francisco. Disponível em: http://www.techonline.com/learning/techpaper/193103703. Acesso em: 22 nov 2007.

DE MICHELI, G. Computer-aided hardware-software codesign, **IEEE Micro**, v. 14, p. 10-16, 1994.

DIAS, F. G. **Conceitos Fundamentais de SRPLV**. Disponível em: http://www.dca.fee.unicamp.br/~gaiotto/projects/srplv/conceitos.html. Acesso em: 14 nov. 2007.

DIAS, F. G.; LOTUFO, R. A. **Melhorias para Sistemas de Reconhecimento da Placa de Licenciamento Veicular**, IV Workshop de Teses e Dissertações em Computação Gráfica e Processamento de Imagens (WTDCGPI 2005), 2005.

DIGILENT, **Componente UART**. Disponível em: http://www.digilentinc.com/Data/VHDLSource/Serial_UART/UART_Component.zip. Acesso em 04 jun 2006.

DRAGHICI, S. A neural network based artificial vision system for license plate recognition. **International Journal of Neural Systems**, v. 8, p. 113-12, 1997.

EMBEDDED SYSTEM. Wikipedia. Disponível em: http://en.wikipedia.org/wiki/Embedded_system. Acesso em: 23 nov 2007.

EROKHIN, V. V. **Single Clock Unsigned Division Algorithm**. Disponível em: http://www.opencores.org/projects.cgi/web/single_clock_divider/overview. Acesso em: 23 ago 2006.

FLETCHER, B. H. **FPGA Embedded Processors** – Revealing the True System Performance, 2005 Embedded Systems Conference, San Francisco. Disponível em: http://www.techonline.com/learning/techpaper/193103682. Acesso em: 22 nov 2007.

FONS, M.; FONS, F.; CANTO, E. **Design of an Embedded Fingerprint Matcher System**, Proceedings of Tenth IEEE International Symposium on Consumer Electronics (ISCE '06), 2006. p.1-6.

GUESUALDI, A. da R.; SEIXAS, J. M. de **Character recognition in car license plates based on principal components and neural processing**, Proceedings of VII Brazilian Symposium on Neural Networks (SBRN 2002), 2002. p. 206-211.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 2nd ed., EUA: Prentice Hall, 2002. 793 p.

GUPTA, P. Hardware-software codesign. **IEEE Potentials**, v. 20, p. 31-32, 2002.

HA, S.; LEE, C.; YI, Y.; KWON, S.; JOO, Y. **Hardware-Software Codesign of Multimedia Embedded Systems**: the PeaCE Approach, Proceedings of 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2006. p. 207-214.

HAYKIN, S. **Neural networks**: a comprehensive foundation. 2nd ed., Upper Saddle River: Prentice Hall, 1999. 842 p.

HEATH, S. **Embedded Systems Design**. 2nd ed., EUA: Newnes, 2003. p. 1-14.

HERVEILLE, R. **Hardware Division Units**. Disponível em: http://www.opencores.org/projects.cgi/web/divider/overview. Acesso em: 23 ago 2006.

HI-TECH SOLUTIONS, **License Plate Recognition** – A Tutorial. Disponível em: http://www.licenseplaterecognition.com. Acesso em: 14 nov. 2007.

HWANG, I.; KANG, B.; GERARD, J. High-resolution image scaler using interpolation filter for multimedia video applications. **IEEE Transactions on Consumer Electronics**, v. 43, p. 813-818, 1997.

IBM **OPB Bus Functional Model Toolkit**: User's Manual – Version 3.5. Disponível em: http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF7785256991006698D3/$file/OpbToolkit.pdf. Acesso em: 29 mar 2007a.

__ **Processor Local Bus Functional Model Toolkit**: User's Manual – Version 4.9.2. Disponível em: http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF778525699100664DAB/$file/PlbToolkit.pdf. Acesso em: 29 mar 2007b.

IÑIGO, R. M. Application of machine vision to traffic monitoring and control. **IEEE Transactions on Vehicular Technology**, v. 38, p. 112-122, 1989.

__ Traffic Monitoring and Control Using Machine Vision: A Survey. **IEEE Transactions on Industrial Electronics**, v. IE-32, p. 177-185, 1985.

JÄHNE, B.; HAUßECKER, H. **Computer Vision and Applications** – A Guide for Students and Practitioners. EUA: Academic Press, 2000. p. 578-578.

JEAN, S.-D.; LIU, C.-M.; CHANG, C.-C.; CHEN, Z. **A new algorithm and its VLSI architecture design for connected component labeling**, IEEE International Symposium on Circuits and Systems (ISCAS '94), 1994. v. 2, p. 565-568.

KIM, C.-H.; SEONG, S.-M.; LEE, J.-A.; KIM, L.-S. Winscale: an image-scaling algorithm using an area pixel model. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 13, p. 549-553, 2003.

KÖLSCH, M.; KISACANIN, B. **Embedded Computer Vision and Smart Cameras**, 2005 Embedded Systems Conference, San Francisco. Disponível em: http://www.techonline.com/learning/techpaper/199203583. Acesso em: 03 jun 2007.

KUTTNER, C. Hardware-Software Codesign Using Processor Synthesis. **IEEE Design & Test of Computers**, v. 13, p. 43-53, 1996.

KWASNICKA, H.; WAWRZYNIAK, B. **License Plate Localization and Recognition in Camera Pictures**, 3rd Symposium on Methods of Artificial Intelligence (AI METH 2002), 2002.

LEEDHAM, G.; YAN, C.; TAKRU, K.; TAN, J. H. N.; MIAN, L. **Comparison of some thresholding algorithms for text/background segmentation in difficult document images**, Proceedings of 7th International Conference on Document Analysis and Recognition, 2003. p. 859-864.

MARTÍN, F.; GARCÍA, M.; ALBA, J. L. **New Methods for Automatic Reading of VLP's (Vehicle License Plates)**, Signal Processing Pattern Recognition and Applications (SPPRA 2002), 2002.

MATLAB, **Fixed-Point C Development.** Demonstração referente ao pacote de ponto fixo disponibilizado pelo ambiente MATLAB, v. 7.2.0.232 (R2006a), 2006.

MOLZ, R. F. **Uma Metodologia para o Desenvolvimento de Aplicações de Visão Computacional utilizando um projeto conjunto de Hardware e Software**. 2001, 80f. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

NARASIMHAN, N.; SRINIVASAN, V.; VOOTUKURU, M.; WALRATH, J.; GOVINDARAJAN, S.; VEMURI, R. **Rapid prototyping of reconfigurable coprocessors**, Proceedings of International Conference on Application Specific Systems, Architectures and Processors (ASAP 96), 1996. p. 303-312.

NN FAQ, **Neural Network FAQ**. Disponível em: ftp://ftp.sas.com/pub/neural/FAQ.html#A_kinds. Acesso em 01 abr 2008.

NI **NI Vision Concepts Manual,** EUA: National Instruments Corporation, 2007. cap. 1, p. 1-5.

NICOL, C. J. A Systolic Approach for Real Time Connected Component Labeling. **Computer Vision and Image Understanding**, v. 61, p. 17-31, 1995.

NIEMANN, R. **Hardware/Software Codesign**. Disponível em: http://ls12-www.cs.uni-dortmund.de/~niemann/codesign/codesign.html. Acesso em: 23 nov 2007.

NIJHUIS, J. A. G.; TER BRUGGE, M. H.; HELMHOLT, K. A.; PLUIM, J. P. W.; SPAANENBURG, L.; VENEMA, R. S.; WESTENBERG, M. A. **Car license plate recognition with neural networks and fuzzy logic**, Proceedings of IEEE International Conference on Neural Networks, 1995. v. 5, p. 2232-2236.

OU, J.; PRASANNA, V. K. **MATLAB/Simulink Based Hardware/Software Co-Simulation for Designing Using FPGA Configured Soft Processors**, Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, 2005. p. 148b-148b.

OWENS, R. **Geometric Transformations** Material referente á disciplina de Visão Computacional. Disponível em: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT5/node5.html. Acesso em: 1 dez 2007.

PACHECO, M. A. **Nova Metodologia de Localização de Regiões Candidatas em Imagens Digitais Utilizando Arquiteturas Reconfiguráveis**. 2007. 95f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Santa Maria, Santa Maria, 2007.

PHOTOCOP **Plate Recognition**. Disponível em:, http://www.photocop.com/recognition.htm. Acesso em: 14 nov 2007.

POLIDÓRIO, A.M. & BORGES, D.L. **Um Método de Reconhecimento Sintático de Caracteres para Identificação de Placas de Veículos**, Anais do X Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (SIBGRAPI'97), 1997.

PRATT, W. K. **Digital Image Processing**, 4. ed. EUA: Wiley-Interscience, 2007a. cap. 13, p. 387-417.

PRATT, B. **Simple Speedups for any ML-XUP Virtex-II Pro EDK System**. Disponível em: http://www.et.byu.edu/groups/ececmpsysweb/cmpsys.2005.winter/teams/ibox/groups/ibox/public/brian/index.html. Acesso em: 17 jun 2007b.

QUERCUS TECHNOLOGIES **The Automatic Number Plate Recognition Tutorial**. Disponível em: http://www.anpr-tutorial.com. Acesso em: 14 nov. 2007.

RASQUINHA, A. & RANGANATHAN, N. **C^3**: a chip for connected component labeling, Proceedings of Tenth International Conference on VLSI Design, 1997. p. 446-450.

ROSENFELD, A.; PFALTZ, J. L. Sequential Operations in Digital Picture Processing. **Journal of Association for Computing Machinery**, v. 13, p. 471-494, 1966.

RUSS, J. C. **The Image Processing Handbook**, 5 ed. EUA: CRC Press, 2007. cap. 1, p. 1-76.

SONKA, M.; HLAVAC, V.; BOYLE, R.. **Image Processing, Analysis, and Machine Vision**, 3. ed. EUA: Thomson Learning, 2007. 829 p.

SOUZA, F.P.C.de **Localização e Leitura Automática de Caracteres Alfanuméricos** – Uma Aplicação na Identificação de Veículos, 2000. 96f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000.

TIAN, H.; LAM, S. K.; SRIKANTHAN, T. Area-time efficient between-class variance module for adaptive segmentation process, **IEE Proceedings – Vision, Image and Signal Processing**, v. 150, p. 263-269, 2003.

TOWNSEND, A. When Hardware Met Software. **Xcell Journal**, Issue 54, p. 6-9, Third Quarter, 2005.

TURNER, S. **Advanced techniques for travel time data collection**, Proceedings of 6th International Vehicle Navigation and Information Systems Conference (VNIS 1995), 1995. p. 40-47.

WANG, K.-B.; CHIA, T. L.; CHEN, Z.; LOU, D.-C. Parallel Execution of a Connected Component Labeling Operation on a Linear Array Architecture. **Journal of Information Science and Engineering**, v. 19, p. 353-370, 2003.

WOLF, W. A Decade of Hardware/Software Codesign. **Computer**, v. 36, p. 38-43, 2003.

XILINX **Embedded System Tools Reference Manual**. Documentação do ambiente EDK 8.2i, 2006a. p. 100-131.

XILINX **Virtex-II Pro and Virtex-II Pro X Platform FPGAs**: Complete Data Sheet, DS083 (v4.6). Disponível em: http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf. Acesso em: 12 jun 2007.

XILINX **XPS Help Documentation:** seção *Creating New Peripherals*. Documentação do ambiente EDK 8.2i, 2006b.

XILINX **XPS Help Documentation**: seção *Software Development Overview*. Documentação do ambiente EDK 8.2i, 2006c.

XILINX **XPS Help Documentation**: seção *Using the Base System Builder*. Documentação do ambiente EDK 8.2i, 2006d.

XITE **X-based Image Processing Tools and Environment**, University of Oslo, Denmark. Disponível em: http://www.ifi.uio.no/~blab/Software/Xite/. Acesso em: 26 set 2005.

YANG, S.-W.; SHEU, M.-H.; WU, H.-H.; CHIEN, H.-E.; WENG, P.-K.; WU, Y.-Y **VLSI architecture design for a fast parallel label assignment in binary image**, 2005 IEEE International Symposium on Circuits and Systems (ISCAS 2005), 2005. v. 3, p. 2393-2396.

YAP, K. S.; TAY, Y. H.; KHALID, M.; AHMAD, T. **Vehicle Licence Plate Recognition by Fuzzy Artmap Neural Network**, 1999 World Engineering Congress (WEC99), 1999.

ANEXO A – Lista de publicações

DESSBESELL, G. F.; PACHECO, M. A.; MARTINS, J. B. dos S.; MOLZ, R. F. **An Area Cost Optimized Fast Parallel Label Assignment VLSI Architecture**, Proceedings of 3rd Southern Conference on Programmable Logic (SPL 2007), Mar del Plata, Argentina, 26-28 fev 2007. p. 155-160.

DESSBESELL, G. F.; PACHECO, M. A.; MARTINS, J. B. dos S.; MOLZ, R. F. **A VLSI Architecture Suitable for Mid-Level Image Processing**, artigo aceito para apresentação oral no 4th Southern Conference on Programmable Logic (SPL 2008), San Carlos de Bariloche, Argentina, 26-28 mar 2008. p. 87-92.