Universidade de São Paulo
Instituto de Física de São Carlos

Paulo Matias

Discriminação em tempo real de sinais de peixes elétricos pulsadores usando FPGAs

São Carlos

2015

Paulo Matias

Discriminação em tempo real de sinais de peixes elétricos pulsadores usando FPGAs

Tese apresentada ao Programa de Pós-Graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo, para obtenção do título de doutor em Ciências.

Área de Concentração: Física Aplicada – Opção: Física Computacional

Orientador: Prof. Dr. Jan Frans Willem Slaets

Versão Corrigida
(versão original disponível na Unidade que aloja o Programa)

São Carlos
2015

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica revisada pelo Serviço de Biblioteca e Informação do IFSC,
com os dados fornecidos pelo(a) autor(a)

Matias, Paulo
Discriminação em tempo real de sinais de peixes elétricos pulsadores usando FPGAs / Paulo Matias; orientador Jan Frans Willem Slaets - versão corrigida -- São Carlos, 2015.
121 p.

Tese (Doutorado - Programa de Pós-Graduação em Física Computacional) -- Instituto de Física de São Carlos, Universidade de São Paulo, 2015.

1. Neurobiofísica. 2. Peixes elétricos pulsadores. 3. Instrumentação eletrônica. 4. FPGA. 5. Tempo real. I. Slaets, Jan Frans Willem, orient. II. Título.

RESUMO

MATIAS, P. *Discriminação em tempo real de sinais de peixes elétricos pulsadores usando FPGAs*. 2015. 121 p. Tese (Doutorado em Ciências) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2015.

Peixes elétricos de campo fraco comunicam-se por meio de descargas elétricas de forma de onda estereotipada, variando somente o intervalo entre pulsos de acordo com a informação a ser transmitida. Esse mecanismo de codificação é similar ao encontrado em diversos circuitos de neurônios conhecidos, o que torna esses animais excelentes modelos para o estudo de sistemas de comunicação naturais, permitindo experimentos que envolvem tanto aspectos comportamentais como neuroetológicos. É um desafio realizar análises de dados coletados de mais de um peixe nadando livremente, pois os padrões de descargas de órgão elétrico (DOE) dependem da posição dos animais e de suas orientações com relação aos eletrodos de medida. Contudo, como cada peixe emite uma forma de onda de DOE característica, ferramentas computacionais podem ser empregadas para associar cada DOE ao respectivo peixe. Neste trabalho, descrevemos um método computacional capaz de reconhecer DOEs de pares de peixes usando vetores de características normalizados, obtidos aplicando a transformada de Fourier e a transformada complexa de dupla árvore de pacote *wavelet*. Empregamos máquinas de vetores de suporte como classificadores, e um algoritmo de regra de continuidade permite resolver problemas causados por DOEs sobrepostas e saturação de sinais. Procedimentos de validação com *Gymnotus* sp. mostraram que as DOEs podem ser atribuídas corretamente a cada peixe com apenas dois erros por milhão de descargas. Para permitir que esse processo de discriminação ocorra em tempo real, implementamos uma arquitetura de hardware dedicada e maciçamente paralela em um *field programmable gate array* (FPGA) para executar a etapa de maior esforço computacional do algoritmo de discriminação. Como resultado, obtivemos um sistema híbrido de hardware e software de tempo real que foi capaz de atender a um requisito

de latência máxima de 1 ms, o que permite mimetizar o tempo de resposta de importantes sistemas sensoriais elétricos de *Gymnotus* sp. Com o auxílio de nossa instrumentação, diversos experimentos com realimentação poderão ser propostos, permitindo que um modelo computacional interaja com dois peixes em uma preparação *in vivo* naturalística.

PALAVRAS-CHAVE: Neurobiofísica. Peixes elétricos pulsadores. Instrumentação eletrônica. FPGA. Tempo real.

ABSTRACT

MATIAS, P. *Real-time discrimination of pulse-type electric fish signals using FPGAs*. 2015. 121 p. Tese (Doutorado em Ciências) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2015.

Pulse-type weakly electric fishes communicate through electrical discharges with a stereotyped waveform, varying solely the interval between pulses according to the information being transmitted. This simple codification mechanism is similar to the one found in various known neuronal circuits, which renders these animals as good models for the study of natural communication systems, allowing experiments involving behavioral and neuroethological aspects. Performing analysis of data collected from more than one freely swimming fish is a challenge since the detected electric organ discharge (EOD) patterns are dependent on each animal's position and orientation relative to the electrodes. However, since each fish emits a characteristic EOD waveform, computational tools can be employed to match each EOD to the respective fish. In this work we describe a computational method able to recognize fish EODs from dyads using normalized feature vectors obtained by applying Fourier and dual-tree complex wavelet packet transforms. We employ support vector machines as classifiers, and a continuity constraint algorithm allows us to solve issues caused by overlapping EODs and signal saturation. Validation procedures with *Gymnotus* sp. showed that EODs can be assigned correctly to each fish with only two errors per million discharges. In order to allow this discrimination process to occur in real-time, we implemented a massively parallel application-specific hardware architecture in a *field programmable gate array* (FPGA) to run the discrimination algorithm step whose computational effort is the highest among the others. As a result, we obtained a hardware and real-time software co-design that was able to meet a maximum latency require-

ment of 1 ms, allowing it to mimic the response time of major electrical sensory systems of *Gymnotus* sp. Our instrumentation will enable the proposal of several novel experiments with closed-loop feedback, allowing a computer model to interact with two fish in a naturalistic *in vivo* preparation.

KEYWORDS: Neurobiophysics. Pulse-type electric fish. Electronic instrumentation. FPGA. Real-time.

Lista de Figuras

Figura 1.1 - Peixes elétricos pulsadores produzem descargas de órgão elétrico (DOEs) estereotipadas, similares aos potenciais de ação de *spiking neurons*. Essas descargas podem ser observadas medindo-se a diferença de potencial elétrico entre dois pontos da água. Uma das grandezas mais importantes para o estudo da eletrocomunicação é o intervalo entre pulsos (IEP), que mede a diferença entre o instante de ocorrência de um pulso e o anterior. 26

Figura 2.1 - Um par de peixes é colocado em aquário em formato de cubo, contendo 8 eletrodos em contato com a água, ou de bloco retangular, contendo 12 eletrodos. Um dos eletrodos é escolhido como referência (ref.), com respeito ao qual a tensão de todos os outros eletrodos é amplificada diferencialmente com um ganho de 100 vezes. Os 7 (aquário cúbico) ou 11 (aquário em bloco retangular) sinais resultantes são amostrados e coletados a uma taxa entre 45 e 50 kS/s por um sistema de aquisição. . 32

Figura 2.2 - Circuito que condiciona os sinais dos eletrodos do aquário para aquisição, amplificando-os diferencialmente com relação à referência. 32

Figura 2.3 - Um instrumento eletricamente isolante, construído com uma peneira plástica e um tubo de PVC, é utilizado para perturbar o peixe durante o estágio de treino sem introduzir ruídos elétricos no aquário. 34

Figura 3.1 - Visão geral de nosso algoritmo de discriminação de descargas de órgão elétrico. 39

Figura 3.2 - Os sinais $\Delta F_c[\cdot]$, obtidos filtrando e diferenciando os dados experimentais coletados de cada canal c, são elevados ao quadrado e somados para calcular $I[\cdot]$. Quando $I[\cdot]$ ultrapassa um limiar t_d, um segmento de atividade de DOE é detectado. Para estabelecer as fronteiras s (e e) do segmento, procuramos por um período "silencioso" de N_b amostras ocorrendo antes (depois) do sinal cair abaixo do limiar t_b. 41

Figura 3.3 - O histograma de uma componente Fourier ou *wavelet* pode ser usado para estimar a distribuição de probabilidade dos valores dessa componente. A área sobreposta entre os histogramas da componente em peixes distintos é interpretada como uma medida da incerteza de classificação caso apenas essa característica do sinal fosse conhecida. Nossa abordagem é escolher as componentes que apresentem a menor área sobreposta em seus histogramas. 44

Figura 3.4 - Um segmento do sinal $A_c[\cdot]$ pode conter DOEs emitidas por ambos os peixes quase ao mesmo tempo. Para identificar a posição s_j onde a DOE emitida pelo j-ésimo peixe começa dentro do segmento, nós exploramos a propriedade de continuidade das formas de onda de DOE emitidas por cada peixe. A figura é organizada como uma tabela, onde cada coluna contém um gráfico apresentando dados relacionados a um certo canal c do sinal. As primeiras duas linhas mostram os vetores $L_{j,c}$, que contém a última DOE previamente reconhecida como sendo emitida pelo j-ésimo peixe, deslocada para a posição s_j sendo avaliada. Na terceira linha, ambos os vetores deslocados são somados, e os $\sum_j L_{j,c}$ resultantes são comparados aos sinais medidos (A_c). Neste exemplo, não ocorre saturação, portanto não há necessidade de se aplicar a função Ξ definida no texto. Os instantes s_j são escolhidos para minimizar a soma de todas as distâncias ao quadrado, mostradas na quarta linha da figura. 47

Figura 3.5 - Tempo médio de execução de cada etapa do algoritmo de discriminação implementado em software. A etapa de "cálculo dos vetores de características" compreende a obtenção e a normalização das transformadas de Fourier e DT-CWPT de todos os canais de uma janela de sinal. A etapa de "máquina de vetores de suporte" corresponde à execução do classificador SVM. A etapa de "sobreposição de disparos anteriores", que domina o tempo de execução, refere-se à aplicação do método de imposição da condição de continuidade. 51

Figura 3.6 - Histograma do tempo de execução da etapa de "sobreposição de disparos anteriores". Além de ser a etapa do software que consome o maior tempo de execução, ocorre uma grande variabilidade desse tempo, prejudicando a utilização desse módulo do software em um sistema capaz de gerar resultados em tempo real. 51

Figura 4.1 - Medida intracraniana do núcleo magnocelular do cérebro do peixe. À esquerda, montagem experimental (Instituto de Investigaciones Biológicas Clemente Estable, Uruguai). À direita, tela do osciloscópio mostrando: no canal 1, sinal coletado do núcleo magnocelular; no canal 2, EOD coletada na água. Existe um atraso de cerca de 1,2 ms entre o término da EOD e o sinal ser observado no núcleo magnocelular. 53

Figura 4.2 - Visão geral do módulo de hardware principal. Os retângulos coloridos dentro do quadro cinza representam submódulos, e as setas indicam o fluxo de dados entre os diferentes módulos. Os retângulos posicionados sobre as bordas do quadro são as interfaces do módulo principal com o mundo exterior. As setas mais claras são os caminhos pelos quais circulam os comandos disponibilizados ao software de tempo real para controlar o hardware. Esses comandos são gerenciados pela regra `handleCmd`, com base em requisições recebidas pela interface `AvalonSlave`. 57

Figura 4.3 - Conexões configuradas no Altera Qsys para permitir que o módulo de hardware principal escrito em Bluespec tenha acesso ao barramento PCI Express. A interface `AvalonSlave` do módulo, utilizada para receber comandos dos módulos de *kernel* e ferramentas de configuração, é conectada ao `bar0`. A interface `AvalonMaster`, por meio da qual o hardware pode escrever diretamente na memória RAM do computador (DMA), é conectada ao `txs`. A interface `InterruptSender`, utilizada pelo hardware para disparar interrupções a serem atendidas por uma rotina registrada por algum módulo de *kernel*, é ligada ao `rxm_irq`. 61

Figura 4.4 - Histograma da latência de atendimento a interrupções do barramento PCI Express (em verde) e do porto paralelo (em vermelho) com RTAI, produzidas por um hardware externo como resposta a um comando enviado por uma tarefa periódica no domínio de tempo real. As medidas foram retiradas do mesmo computador utilizado para a confecção do discriminador. 63

Figura 4.5 - Painel do sistema de aquisição customizado construído como parte desta tese. 64

Figura 4.6 - Fabricação da placa de circuito impresso para o conversor MAX1280. Foi possível utilizar o processo de transferência térmica de toner apesar do circuito integrado ter encapsulamento TSSOP com espaçamento de somente 0,65 mm entre os terminais. 65

Figura 4.7 - Diagrama do circuito de *front-end* analógico do sistema de aquisição customizado. O circuito divide a tensão de entrada por $^{940}/_{120} \approx 7{,}83$ (ponto a), acopla em corrente alternada com valor de repouso em $^{1}/_{2}V_{\text{+LIM}}$ (ponto b), limita a tensão resultante entre $-0{,}3$ V e $V_{\text{+LIM}} + 0{,}3$ V (ponto c) e, por fim, reforça o sinal com um seguidor. 66

Figura 4.8 - Simulação SPICE do *front-end* analógico com $V_{\text{+LIM}} = 2{,}5$ V. Mesmo quando a tensão de entrada (em azul) ultrapassa os 10 V previstos, o circuito é capaz de proteger o conversor AD, mantendo a tensão de saída (em verde) sempre dentro da região tolerada (em vermelho). 66

Figura 4.9 - Circuito lógico de uma fila (FIFO – *first-in first-out*) sincronizadora da biblioteca padrão do Bluespec. Os dados são inseridos na fila em um domínio de *clock* e retirados em outro. O circuito implementa os cuidados necessários para com os ponteiros de cabeça e de cauda da fila, que são sincronizados por cadeias de 2 registradores seguidos (para evitar metaestabilidade) e utilizam um código Gray para impedir que atrasos diferentes na propagação de cada bit do contador corrompam seu valor ao cruzar os domínios de *clock*. 68

Figura 4.10 - Demonstração do modo de aquisição contínua de dados, que permite que o sistema seja utilizado como se fosse uma placa de aquisição comum, monitorando qualquer um dos 16 canais de entrada, mesmo que nem todos estejam habilitados para uso pelo discriminador. 69

Figura 4.11 - Soma dos módulos da transformada de Hilbert (aproximada com filtro IIR) de uma DOE em 4 canais, para um sinal bem-formado A_c e para uma versão saturada B_c da mesma DOE. Sabe-se que o pico da soma de transformadas de Hilbert cai um instante de referência aproximadamente fixo dentro das DOEs de um peixe. Esse comportamento do pico mantém-se resiliente à saturação de um dos canais. 71

Figura 4.12 - Diagrama simplificado (*data-path*) da arquitetura de hardware do módulo de sobreposição de disparos (`DistMinimizer`). As DOEs previamente emitidas pelos peixes A e B são fornecidas ao circuito pelas filas `spkA` e `spkB`. O segmento de sinal atualmente em análise é fornecido pela fila `segm`. Uma vez alimentadas as cadeias de L_{w} registradores, os multiplexadores conectados às filas alternam para a entrada superior, e o circuito funciona durante $L_{\mathsf{w}}^2 + \log_2 L_{\mathsf{w}}$ ciclos. Ao longo desse tempo, a cadeia de registradores `spkB` rotaciona as amostras a uma velocidade L_{w} vezes menor que a cadeia `spkA`. Somamos os valores contidos na cadeia `spkA` com os da `spkB`, subtraímos dos valores da cadeia `segm`, e por fim calculamos o módulo. Os valores de módulo passam por uma árvore de somadores organizados em um *pipeline* de $\log_2 L_{\mathsf{w}}$ estágios. O valor mínimo encontrado ao longo de todo esse processo é armazenado em um registrador, juntamente com o número do ciclo no qual este foi obtido, o que permite recuperar as posições para as quais os sinais haviam sido deslocados naquele instante. 75

Figura 4.13 - O software de teste e validação tem acesso ao hardware por meio de bibliotecas padronizadas, escritas em linguagem Python, contendo rotinas que mapeiam uma-a-uma para as operações correspondentes nos blocos de hardware, descritos em Bluespec. A comunicação entre o computador e o FPGA é realizada por meio do barramento JTAG, acessível na maioria dos kits de desenvolvimento através de uma interface USB. A ferramenta `quartus_stp` faz a interface entre o software em Python e o barramento JTAG. No lado do hardware, o bloco `sld_hub` distribui a comunicação JTAG para os diversos circuitos que compõe as bibliotecas de hardware. . 78

Figura 4.14 - Exemplo de uso das bibliotecas JTAG para depuração do hardware desenvolvido nesta tese. Á esquerda, apresentamos o código do hardware (em Bluespec) e, à direita, o código do software (em Python). Para interceptar os resultados do módulo `mkWindowDMABuffer`, inserimos um módulo `mkTap` em sua saída, que insere todo dado trafegado por esse caminho na fila de 4096 elementos associada ao ponto de medida ‘‘`TAP0`’’. O software instancia uma conexão com essa fila de saída e lê todo o seu conteúdo em um vetor, salvando os dados coletados em um arquivo ‘‘`tap0.pickle`’’. 80

Figura 4.15 - Geramos aleatoriamente dois sinais (A e B), que permitimos que saturem em alguns pontos, além de um ruído de amplitude 2 a 4× superior àquela observada nos experimentos. O software de testes escolhe duas posições aleatórias, mostradas no gráfico como barras verticais cinzas, que passam a ser o novo início dos sinais A e B quando estes são deslocados. A soma dos sinais deslocados e do ruído produz o sinal mostrado no gráfico mais abaixo. Enviamos ao FPGA somente os sinais A e B originais (antes de serem deslocados) e o sinal de soma, e verificamos que o circuito é capaz de recuperar as posições de deslocamento. 81

Figura 4.16 - Otimização do classificador SVM com o uso de instruções vetoriais da família AVX. O cálculo pode ser realizado em duas etapas — primeiramente são obtidos todos os quadrados das distâncias entre vetores, e apenas em seguida é aplicado o núcleo RBF. Ambas as etapas podem ser paralelizadas em vetores de 4 elementos. 83

Figura 4.17 - A transformada *wavelet* (DT-CWPT), apesar de não ter sido manualmente otimizada com instruções AVX, faz algum uso destas devido à vetorização automática do compilador GCC. As instruções `vmulps` e `vaddps` são pertencentes ao conjunto AVX e, respectivamente, multiplicam ou somam um vetor de 8 números de ponto flutuante de precisão simples. . 85

Figura 4.18 - Funções e macros modificadas no arquivo `rtai_fpu_64.h` para possibilitar o uso de instruções AVX em espaço de *kernel* nas rotinas de tempo real. Substituímos a instrução `fxsave`, que salva apenas o estado utilizado por instruções x87, MMX, SSE e SSE2, pela `xsave`, capaz de salvar o estado relacionado a quaisquer extensões suportadas pelo processador. . 87

Figura 5.1 - Curvas de característica de operação do receptor (COR) comparando os métodos de SVM e de correlação cruzada ao classificar janelas contendo um único DOE emitido por um indivíduo de um par (peixes A e B). A taxa de verdadeiros positivos denota a razão de DOEs emitidos pelo peixe A que foram identificados corretamente. De forma similar, a taxa de falsos positivos corresponde à razão dos DOEs emitidos pelo peixe B que foram identificados incorretamente. Os gráficos foram traçados para dois pares de peixes distintos — aqueles que obtiveram a melhor e a pior acurácia de classificação, respectivamente, dentre as 6 díades usadas nos experimentos. 90

Figura 5.2 - Gráficos de intervalo entre pulsos (IEP) apresentando dados provenientes de dois algoritmos distintos de discriminação, aos quais foram fornecidos os mesmos dados, coletados de um par de peixes que nadavam livremente. A sub-figura (A) mostra resultados de uma versão de nosso algoritmo que usava somente a SVM (sem se aproveitar da propriedade de continuidade das formas de onda), e portanto apresentava dificuldades quando ambos os peixes emitiam DOEs quase ao mesmo tempo ou quando algum canal de medida saturava. Os erros de discriminação podem ser imediatamente observados no gráfico: a região (i) contém DOEs não-detectadas, e está localizada aproximadamente no dobro do IEP médio; a região (ii) contém falsos positivos, e apresenta IEPs abaixo do patamar médio. A sub-figura (B) mostra resultados de nosso algoritmo implementado por completo, como descrito neste trabalho. Nenhum erro pode ser apontado neste gráfico de IEP. 91

Figura 5.3 - Tempo de execução e *speedup* de diferentes implementações do módulo de sobreposição de disparos, variando-se o número de canais e mantendo-se fixa uma janela de 128 amostras. Enquanto o hardware executa em tempo praticamente constante (135 μs), o tempo de execução da implementação em software cresce linearmente, e em torno de 3 canais já não é mais aceitável para uma implementação de tempo real. O *speedup* do hardware é praticamente linear com o número de canais, enquanto que uma implementação em software *multi-thread* alcança um *speedup* aproximadamente constante de 1,9$\times$. 93

Figura 5.4 - Tempo de execução e *speedup* de diferentes implementações do módulo de sobreposição de disparos, variando-se o número de amostras da janela e processando-se sempre um único canal. Os tempos de execução do hardware e do software foram ajustados, respectivamente, a polinômios do segundo e do terceiro grau, por se tratarem de algoritmos $\Theta(L_{\mathsf{w}}^2)$ e $\Theta(L_{\mathsf{w}}^3)$. O software *multi-thread* é mais lento que o *single-thread* para 8 amostras, mas torna-se gradativamente mais rápido aumentando-se a janela, até saturar com *speedup* praticamente ideal (2$\times$ em um computador de 2 núcleos). A implementação em hardware é sempre mais rápida, e tende a um *speedup* linear para janelas grandes ($>$64 amostras). 94

Figura 5.5 - Tempo de execução de diferentes implementações de SVM (apenas classificação), variando-se o número de vetores de suporte e processando-se sempre 11 canais. Para nossa aplicação, a biblioteca `libsvm-dense` é mais rápida que a `libsvm` padrão, entretanto nossa implementação com instruções AVX é sempre superior a ambas as bibliotecas (tanto em suas versões *single* como *multi-thread*), além de apresentar uma dispersão muito menor do tempo de execução. 97

Figura 5.6 - A quantidade de blocos lógicos utilizados no FPGA cresce de forma aproximadamente linear com o número de canais de entrada habilitados, exceto quando a ocupação se aproxima de 8126 LABs (89% dos blocos lógicos). Quando o módulo de sobreposição de disparos é configurado para utilizar uma janela de 256 amostras, o `EP4SGX230` comporta no máximo 8 canais. Ativando-se o filtro decimador na entrada do módulo (que reduz a janela para 128 amostras), é possível habilitar todos os 16 canais oferecidos pelo conversor AD. 99

Figura 5.7 - A quantidade de blocos de memória utilizados no FPGA aumenta proporcionalmente ao número de canais e à menor potência de 2 maior ou igual a este, permanecendo porém sempre muito menor que o limite de 14 625 kbit oferecido pelo `EP4SGX230`. 99

Figura 5.8 - Experimento de discriminação em tempo real das DOEs de uma díade. A tela do computador exibe um gráfico de IEP de ambos os peixes que é atualizado *on-line*, ou seja, ao mesmo tempo em que as DOEs são discriminadas. Mesmo com a carga do sistema inerente a disponibilizar uma interface gráfica interativa, o discriminador é capaz de atender aos requisitos de tempo real. 101

Figura 5.9 - Tela do osciloscópio mostrando a captura de duas DOEs quase consecutivas. O canal 1 (em verde) corresponde ao valor negado da soma dos quadrados dos sinais de todos os 11 canais do aquário. O canal 2 (em vermelho) apresenta nível lógico alto durante todo o período em que a DOE está sendo processada pela rotina de tempo real no RTOS. O canal 3 (em azul) é programado para disparar 2 ms após o ponto de referência (pico da soma de transformadas de Hilbert) de toda DOE que for detectada como proveniente do peixe A. 102

Figura 5.10 - A latência mínima ($\Delta t_{\text{min}} \approx 500\mu$s) e a máxima ($\Delta t_{\text{max}} \approx 950\mu$s) do discriminador são aferidas ativando o modo de persistência do osciloscópio e definindo o canal 3 (em azul) como *trigger* a fim de alinhar os sinais observados com um ponto de referência aproximadamente fixo. 102

Lista de Tabelas

Tabela 5.1 - Acurácia da SVM, estimada por validação cruzada em 10 dobras e por validação simples, comparada com a acurácia do método de correlação cruzada, estimada por validação simples, em 6 díades diferentes. 89

Tabela 5.2 - Quantidade de vetores de suporte da SVM treinada com 6 díades diferentes. 97

Sumário

Capítulo 1 Introdução 25

1.1 Principais contribuições deste trabalho 29

Capítulo 2 Métodos experimentais 31

2.1 Montagem experimental 31

2.2 Procedimento experimental 34

2.3 Procedimento de validação 35

Capítulo 3 Algoritmo de discriminação 37

3.1 Trabalho preexistente: correlação cruzada 37

3.2 Novo estado da arte: algoritmo proposto 39

3.2.1 Segmentação de DOEs 40

3.2.2 Extração de características 42

3.2.3 Seleção de características 43

3.2.4 Classificação supervisionada 45

3.2.5 Condição de continuidade 46

3.3 Identificação de gargalos de desempenho 49

Capítulo 4 Implementação de tempo real 53

4.1 Visão geral da arquitetura 56

4.2 Infraestrutura de comunicação via PCI Express 60

4.3 Sistema de aquisição customizado 64

4.3.1 *Front-end* analógico 65

4.3.2 Módulo de controle dos conversores AD 67

4.3.3 Infraestrutura de teste 69

4.4 Módulo delimitador de janelas 72

4.5 Módulo de transferência de janelas 74

4.6 Módulo de sobreposição de disparos 74

4.7 Ferramentas para teste e depuração de hardware 78

4.7.1 Interceptação de dados trocados entre módulos do *pipeline* 79

4.7.2 Teste do módulo de sobreposição de disparos 81

4.8 Otimização dos módulos em software 82

4.8.1 Implementação da SVM com instruções vetoriais 84

4.8.2 Ajustes finos na implementação de DT-CWPT 85

4.9 Adaptação do RTAI para uso de instruções AVX 86

Capítulo 5 Resultados 89

5.1 Acurácia de classificação 89

5.2 Desempenho da implementação de tempo real 92

5.2.1 Módulo de sobreposição de disparos 95

5.2.2 SVM com instruções AVX 96

5.2.3 Cálculo dos vetores de características 98

5.3 Consumo de recursos de hardware do FPGA 98

5.4 Latência total 100

Capítulo 6 Sugestões para trabalhos futuros 105

6.1 Utilização do sistema 105

6.1.1 Experimentos com realimentação elétrica 105

6.1.2 Experimentos com realimentação luminosa 106

6.2 Implementação de variações do sistema 107

6.2.1 Implementação em kits FPGA de baixo custo 107

6.2.2 Implementação *standalone* em FPGA 108

Capítulo 7 Conclusões 111

Referências 113

Capítulo 1

Introdução

Peixes elétricos pulsadores de campo fraco, tais como os *Gymnotus* sp., são conhecidos por emitir descargas de órgão elétrico (DOEs) estereotipadas, usadas pelos animais para fins de eletrolocalização e eletrocomunicação.[1,2] Nesta tese focaremos somente na eletrocomunicação, aspecto de grande interesse por tratar-se de um mecanismo de troca de informações entre diferentes indivíduos que, apesar de oriundo de uma complexa rede interna de neurônios, funciona com base na emissão de um sinal eletrofisiológico composto por trens de pulsos (similares aos potenciais de ação medidos em *spiking neurons*) que podem ser observados de forma não-invasiva, por meio do potencial elétrico (1 a 5 V) produzido na água. Devido à natureza estereotipada dos sinais elétricos emitidos por esses peixes, as pesquisas em eletrocomunicação geralmente focam[3–5] na análise de grandezas obtidas somente a partir do instante de ocorrência (*timestamp*) das DOEs,[6] por exemplo a diferença entre um instante de ocorrência e o anterior, denominada intervalo entre pulsos (IEP), ilustrada na Figura 1.1.

Gravar sinais emitidos por peixes nadando livremente é uma tarefa desafiadora, mas técnicas tradicionais baseadas em arranjos de eletrodos podem ser empregadas para este fim.[7–9] A ideia é similar à gravação de sinais de neurônios do sistema nervoso central de um animal;[10,11] no entanto, aqui os "neurônios" não estão estacionários e, ainda que lembre a mesma situação, a tarefa é mais complicada que realizar gravações de arranjos de eletrodos que apresentam um movimento de deriva ao longo do tempo.[12] Técnicas que permitam a detecção e a discriminação precisa de DOEs emitidas por pares ou grupos de peixes, causando o menor distúrbio possível no sistema estudado, são ferramentas de extremo valor para a pesquisa em neuroeto-

logia,[4,5,13–15] porque tornam possível estudar uma infinidade de cenários de comunicação social em uma montagem experimental naturalística. Entretanto, atualmente há uma carência de ferramentas computacionais capazes de identificar com precisão o indivíduo que emitiu cada pulso gravado durante esses experimentos.

Como as formas de onda das DOEs variam de um peixe para outro,[16] a princípio seria viável empregar técnicas de aprendizagem de máquina, em uma abordagem similar ao problema de *spike sorting*,[17] que é recorrente na literatura de neurociência. Diversos trabalhos confirmam, em casos específicos, que essas diferenças entre DOEs de animais distintos são observáveis: existem relatos de que peixes provenientes de regiões geográficas distintas apresentam formas de onda diferentes,[18] além de estudos que constatam alterações nas DOEs relacionadas ao processo de crescimento em peixes jovens.[19] Apesar desses trabalhos reforçarem a hipótese de que deve ser possível reconhecer qual animal originou uma DOE a partir de sua forma de onda, não é trivial lidar com sinais de peixes cujos movimentos não estejam restritos, devido à variação das formas de onda causada pelo deslocamento da fonte de sinais (peixe) em relação aos eletrodos de captação.[8,20]

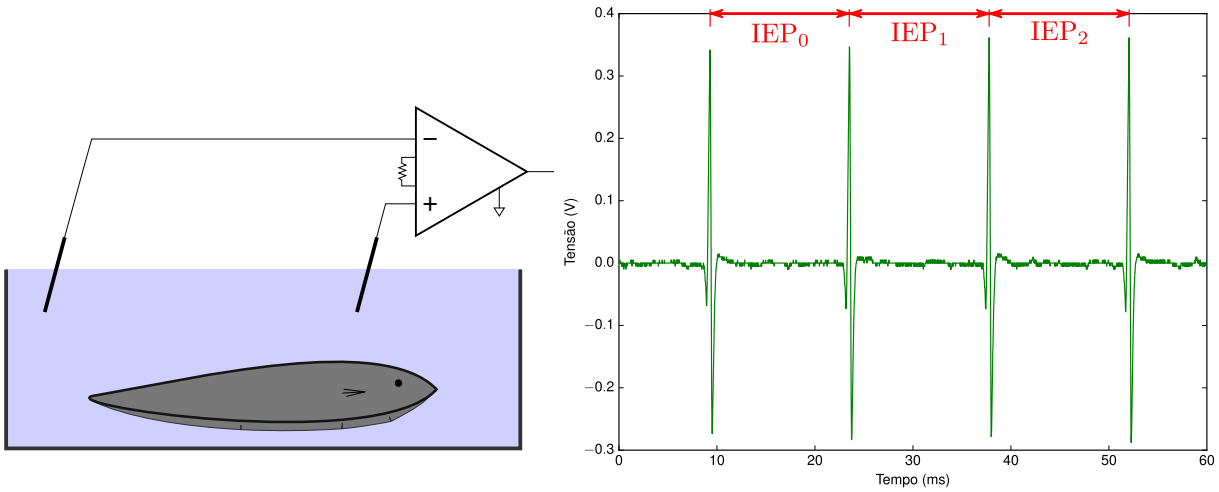

Figura 1.1- Peixes elétricos pulsadores produzem descargas de órgão elétrico (DOEs) estereotipadas, similares aos potenciais de ação de *spiking neurons*. Essas descargas podem ser observadas medindo-se a diferença de potencial elétrico entre dois pontos da água. Uma das grandezas mais importantes para o estudo da eletrocomunicação é o intervalo entre pulsos (IEP), que mede a diferença entre o instante de ocorrência de um pulso e o anterior.

Fonte: Elaborada pelo autor.

Devido a essa dificuldade, a maioria dos trabalhos existentes tenta discriminar os indivíduos empregando procedimentos não-automatizados, que envolvem a inspeção visual da amplitude

e da duração dos pulsos de DOE,[15] algumas vezes com o auxílio de gravações de vídeo que permitem inferir a posição do peixe, que é então correlacionada com mudanças de amplitude e de polaridade dos pulsos.[5] Porém, esses métodos consomem muito tempo do pesquisador, e só podem ser utilizados para obter no máximo poucos minutos de dados experimentais, produzindo portanto uma quantidade de dados de entrada insuficiente para aplicar abordagens estatísticas e de teoria da informação,[9] comumente utilizadas no estudo da codificação de trens de pulsos neurais.[21,22]

O estado da arte anterior a nosso trabalho[4] consistia em um método que primeiro armazenava dois gabaritos (*templates*) de DOE, cada um proveniente de um peixe medido sozinho, e depois calculava a correlação cruzada entre cada DOE capturada em um experimento com uma díade (par de animais) e cada um dos gabaritos. Entretanto, essa abordagem apresentava problemas quando duas DOEs possuíam duração similar ou quando ambos os peixes disparavam pulsos quase ao mesmo tempo, ou seja, quando ocorria uma sobreposição de formas de onda dos dois indivíduos.

Neste trabalho, introduzimos um método computacional que supera as limitações que acabamos de mencionar, sendo capaz de discriminar com acurácia peixes pertencentes a díades. São fornecidas como entradas para esse método medidas de tensão de eletrodos posicionados em locais fixos dentro de um aquário. Aplicamos, essencialmente, um algoritmo de dois passos. Primeiro, porções do sinal com fortes indícios de conterem uma DOE emitida por um único peixe são classificadas por uma máquina de vetores de suporte,[23] com base em um vetor de características normalizado, obtido pela aplicação da transformada de Fourier e da transformada complexa de dupla árvore de pacote *wavelet*.[24,25] Em seguida, exploramos o fato de que as formas de onda variam como uma função contínua da posição durante o movimento de um peixe, o que nos permite encontrar DOEs dentro de segmentos de sinais que possam conter descargas de ambos os peixes.

O capítulo 2 desta tese descreve a montagem e o procedimento experimental necessários à aplicação de nosso método de discriminação, além de discutir um procedimento para validá-lo. Em seguida, o capítulo 3 detalha as técnicas utilizadas pelo método.

No entanto, constatamos que uma implementação de nosso método puramente em software

não seria capaz de atingir desempenho suficiente para cumprir requisitos de tempo real, ou seja, não conseguiria fornecer resultados de reconhecimento das DOEs com baixa latência e baixa variabilidade do tempo de resposta (*jitter*). Sistemas de instrumentação capazes de alcançar requisitos de tempo real são fundamentais para o desenvolvimento de métodos experimentais inovadores em neurociência.[26,27] Mesmo a técnica de *dynamic clamp*,[28] método de grande importância histórica, só pôde avançar graças aos desenvolvimentos na área de computação de tempo real.[29] Trabalhos recentes[30] têm utilizado sistemas operacionais de tempo real (RTOS, do inglês *real-time operating system*) para construir experimentos em *closed-loop* com peixes elétricos, nos quais o instante de ocorrência das DOEs emitidas pelo animal é utilizado para determinar quais estímulos elétricos são produzidos, realimentando o sistema peixe–computador. Entretanto, esses experimentos ainda são limitados à interação do computador com um único animal, devido à inexistência, até então, de um método capaz de discriminar o peixe de origem das DOEs em tempo real.

No capítulo 4, visando preencher também essa lacuna, adotamos uma abordagem de co-projeto de hardware/software[31] para desenvolver um discriminador com latência máxima de 1 ms, tendo em vista mimetizar a latência do sistema sensorial elétrico[32] natural de *Gymnotus* sp. (1 a 2 ms). Para alcançar esse requisito de latência, implementamos uma arquitetura de hardware maciçamente paralela dedicada ao problema de impor uma condição de continuidade à forma de onda das DOEs, etapa essencial para lidar com a existência de disparos quase simultâneos (DOEs sobrepostas) e que é gargalo na implementação em software. As demais etapas de nosso método, que consistem no cálculo dos vetores de características e na classificação por uma máquina de vetores de suporte, foram mantidas em software, porém otimizadas manualmente com o uso do conjunto de instruções vetoriais Intel AVX[33] e adaptadas para executar em espaço de *kernel* no RTAI,[34] um RTOS livre capaz de executar lado a lado com o Linux no mesmo processador.

Os módulos de hardware foram implementados em Bluespec,[35] uma linguagem de descrição de hardware de alto nível com paradigma funcional e de ações atômicas com guardas (GAA, do inglês *guarded atomic actions*).[36] O projeto foi sintetizado em um arranjo de portas programável em campo (FPGA, do inglês *field-programmable gate array*) integrado a um

sistema de aquisição customizado de 16 canais e ao barramento PCI Express do computador.

No capítulo 5, discutimos os resultados de acurácia, desempenho, latência e recursos de hardware utilizados pelo sistema. Demonstramos que o método proposto nesta tese é capaz de atingir taxas de erro (aferidas comparando os resultados do algoritmo com medidas diretas de eletrodos afixados aos peixes) de somente duas partes por milhão de DOEs, e que nossa implementação de tempo real é capaz de discriminar quais indivíduos originaram cada DOE em um aquário de 12 eletrodos com uma latência máxima de 1 ms.

Por fim, o capítulo 6 fornece sugestões para trabalhos futuros envolvendo a utilização deste sistema ou o desenvolvimento de variações do mesmo, e o capítulo 7 apresenta as considerações finais.

1.1 Principais contribuições deste trabalho

Principal inovação

- Nosso método de classificação de formas de onda de peixes elétricos pulsadores define um **novo estado da arte**, reduzindo a taxa de erros do método anterior[4] em até:
  - duas ordens de grandeza quando aplicado sobre um único eletrodo, e sem considerar o histórico de disparos anteriores;
  - quatro ordens de grandeza quando aplicado sobre um arranjo de eletrodos, e considerando o histórico de disparos anteriores.
- Este é o **primeiro** sistema capaz de reconhecer **em tempo real** qual peixe emitiu cada DOE em um aquário contendo uma díade (par de animais).

Contribuição como co-projeto hardware/software

Apesar da literatura contar com diversos exemplos de sistemas de hardware/software envolvendo uso de FPGA e de RTOS aplicados à instrumentação em física experimental ou em engenharia,[37–39] este é o primeiro trabalho do gênero que utiliza uma linguagem de descrição de hardware de alto nível com paradigma GAA ou funcional para projetar os módulos de

hardware contidos no FPGA.

Existem alguns poucos trabalhos envolvendo hardware projetado em Bluespec operando em conjunto com software de tempo real, porém são voltados ou à customização de processadores para implementar infraestrutura de apoio ao escalonador do RTOS,[40] ou ao desenvolvimento de hardware externo para detecção de ataques cibernéticos em sistemas de tempo real,[41] ou seja, não são voltados à instrumentação de experimentos científicos nem à implementação de processos de controle. Além disso, esses trabalhos utilizam arquiteturas embarcadas (MicroBlaze e ARM) em vez da arquitetura PC convencional aqui adotada, que no contexto de nosso trabalho tem a vantagem de ser amplamente difundida e de apresentar baixo custo relativo ao poder computacional.

Contribuições técnicas

Este é o primeiro trabalho a ser publicado de que temos notícia que:

- Implementa um classificador SVM com núcleo RBF manualmente otimizado com instruções AVX (subseção 4.8.1).
- Adapta o RTAI para executar AVX e outras instruções vetoriais em espaço de *kernel* (seção 4.9).

Capítulo 2

Métodos experimentais

Este capítulo descreve a visão mais geral possível dos materiais e métodos envolvidos nesta tese, tratando o sistema desenvolvido por nós como uma caixa preta. A seção 2.1 descreve os equipamentos básicos de um laboratório de pesquisa em eletrocomunicação de peixes. Em seguida, a seção 2.2 descreve os requisitos de operação mais essenciais de nosso sistema, ou seja, como espera-se que ele seja utilizado. Por fim, a seção 2.3 descreve uma forma de realizar uma medida direta de acurácia do sistema de discriminação como um todo.

2.1 Montagem experimental

Nosso laboratório dispõe de dois aquários diferentes para a realização de experimentos. O menor deles possui capacidade para 64 litros de água, formando um cubo de 40 cm de lado, com eletrodos em cada um de seus 8 vértices. O maior comporta 250 litros de água, formando um bloco retangular de 100×50×50 cm, com 12 eletrodos dispostos como se o aquário fosse uma justaposição de dois cubos de 50 cm de lado. Na Figura 2.1, o aquário maior é representado em tom mais claro (cinza), como se fosse uma extensão à esquerda do aquário menor.

Ambos os aquários são montados dentro de gaiolas de Faraday para reduzir a indução de ruído elétrico externo. Os eletrodos são fios de aço inoxidável com 0,2 mm de diâmetro inseridos através da cola de silicone nos cantos do aquário, fazendo com que 1 a 2 mm de comprimento de cada fio fique em contato com a água.

Um dos eletrodos é sempre escolhido como referência, com respeito ao qual a tensão

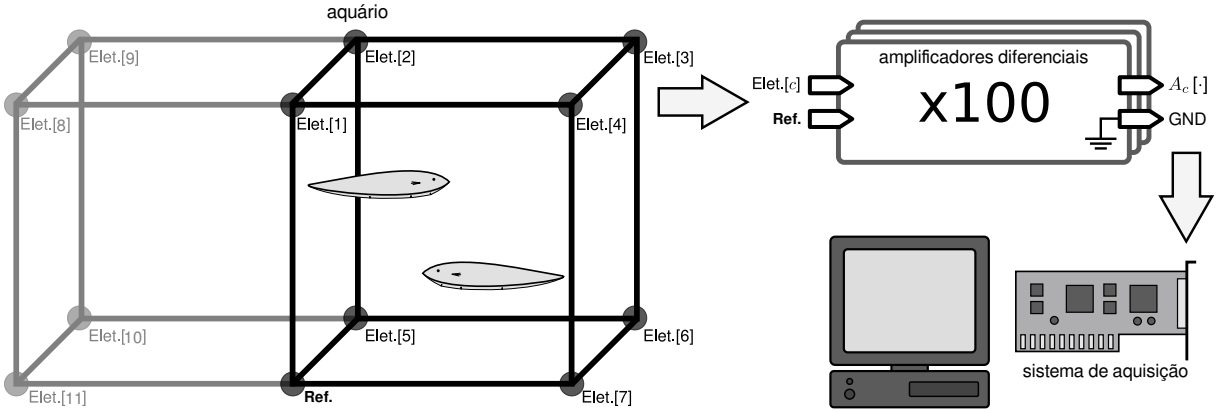

Figura 2.1- Um par de peixes é colocado em aquário em formato de cubo, contendo 8 eletrodos em contato com a água, ou de bloco retangular, contendo 12 eletrodos. Um dos eletrodos é escolhido como referência (ref.), com respeito ao qual a tensão de todos os outros eletrodos é amplificada diferencialmente com um ganho de 100 vezes. Os 7 (aquário cúbico) ou 11 (aquário em bloco retangular) sinais resultantes são amostrados e coletados a uma taxa entre 45 e 50 kS/s por um sistema de aquisição.

Fonte: Elaborada pelo autor.

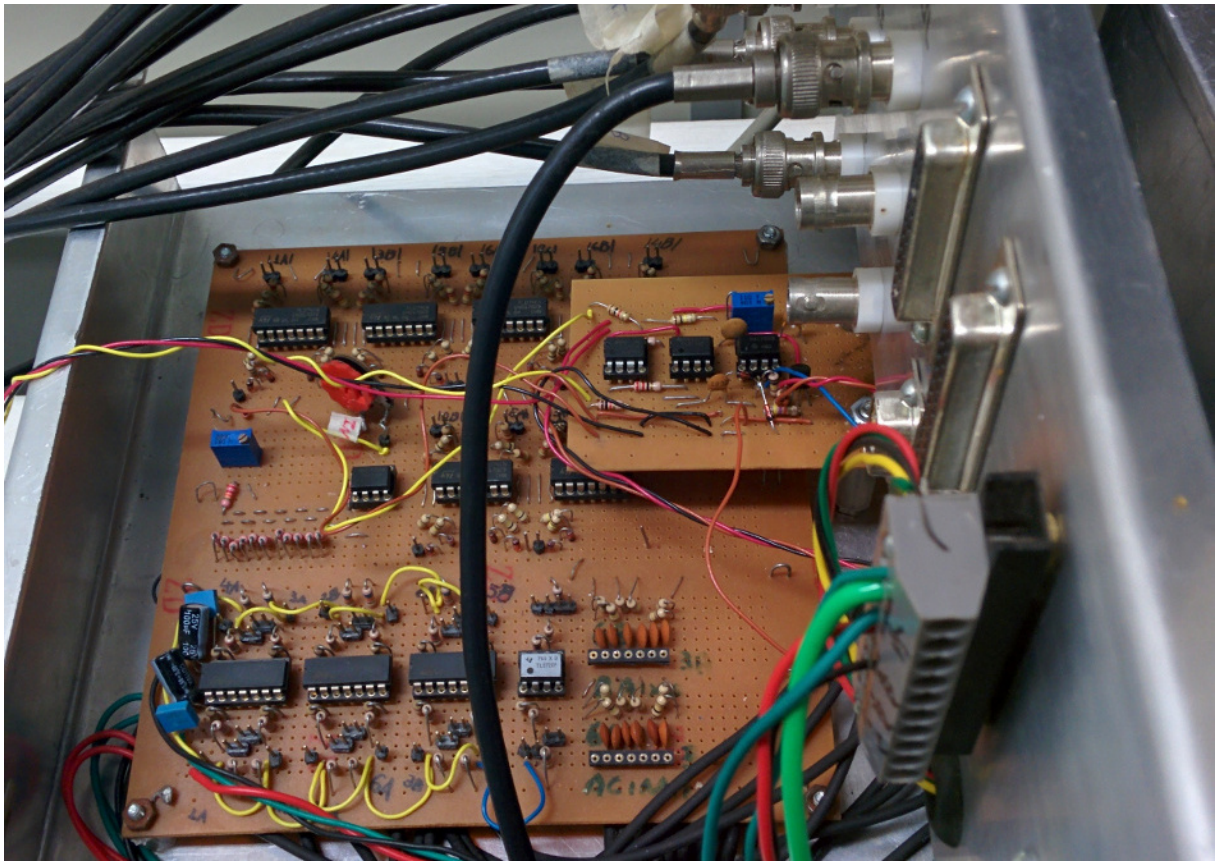

Figura 2.2- Circuito que condiciona os sinais dos eletrodos do aquário para aquisição, amplificando-os diferencialmente com relação à referência.

Fonte: Elaborada pelo autor.

dos outros eletrodos é amplificada diferencialmente 100 vezes, usando amplificadores operacionais LM308.[42] Uma vez amplificados, limitamos os 7 ou 11 sinais resultantes à faixa de -10 a 10 volts. Essa saída é, por fim, direcionada a um sistema de aquisição.

Quando um peixe se aproxima muito do eletrodo de referência, uma tensão excessiva pode ser produzida entre a referência e os outros eletrodos, saturando ao mesmo tempo os sinais amplificados de **todos** eles. Para evitar esse problema, afixamos uma peça de tule de náilon ao vidro do aquário, próximo ao eletrodo de referência, prevenindo a aproximação dos peixes.

A geometria dos aquários e o posicionamento dos eletrodos aqui descrito são os mesmos adotados por Forlim e Pinto.[9] Trata-se de uma montagem fácil de replicar e que fornece uma boa relação sinal-ruído (SNR, do inglês *signal-to-noise ratio*) em pelo menos um dos eletrodos, independente da posição do peixe. Entretanto, nosso algoritmo é bastante geral e nada impede que ele seja utilizado em conjunto com outras geometrias, por exemplo um aquário circular com múltiplos eletrodos de referência como descrito por Jun, Longtin e Maler.[8]

A saída do nosso conjunto de amplificadores (montagem mostrada na Figura 2.2) é eletricamente compatível com diversos sistemas de aquisição comerciais ou customizados, em particular com os três sistemas descritos a seguir, atualmente disponíveis na infraestrutura de nosso laboratório:

- Placa de aquisição **PCI MIO-16-E1** da National Instruments, com 16 canais de entrada analógica, resolução de 12 bits, e taxa de amostragem máxima de 1250 kS/s distribuída entre os canais utilizados. Os sinais podem ser gravados com o software DASYLab para análise *off-line* com o método descrito no capítulo 3. Na prática, utilizamos uma taxa de amostragem de 50 kS/s para cada canal.

- Sistema de aquisição **Digidata 1322A** da Axon Instruments, conectado a um barramento SCSI, com 16 canais de entrada analógica, resolução de 16 bits, e taxa de amostragem máxima de 500 kS/s distribuída entre os canais utilizados. Os sinais podem ser gravados com o software AxoScope para análise *off-line* com o método descrito no capítulo 3. Na prática, utilizamos a maior taxa de amostragem possível (45,45 kS/s) em gravações com 11 canais, e 50 kS/s em gravações com um número menor de canais.

Figura 2.3- Um instrumento eletricamente isolante, construído com uma peneira plástica e um tubo de PVC, é utilizado para perturbar o peixe durante o estágio de treino sem introduzir ruídos elétricos no aquário.

Fonte: Elaborada pelo autor.

- Sistema de aquisição **customizado**, desenvolvido como parte desta tese, contendo dois conversores MAX1280[43] controlados por um FPGA conectado ao barramento PCI Express. Possui 16 canais de entrada analógica, resolução de 12 bits, e taxa de amostragem de 50 kS/s por canal. É utilizado para discriminação em tempo real de DOEs, conforme descrito no capítulo 4.

2.2 Procedimento experimental

Experimentos com uma díade de peixes são efetuados em duas etapas:

- **Estágio de treino**: Nesta etapa, um peixe de cada vez é inserido sozinho no aquário durante alguns minutos, para que as DOEs de cada indivíduo nadando livremente sejam medidas separadamente. Geralmente, 15 minutos são o bastante para coletar cerca de 10^5 DOEs, suficientes para treinar e testar nosso classificador. Durante esse tempo, uma peneira de plástico com cabo estendido por um tubo de PVC (Figura 2.3) pode ser utilizada para perturbar o peixe e fazer com que este realize uma maior variedade de movimentos ao longo de todo o aquário, melhorando a qualidade dos dados de treinamento.

- **Experimento principal**: Consiste em colocar ambos os peixes ao mesmo tempo dentro

do aquário. Os dados adquiridos durante esta etapa podem ser discriminados pelo nosso algoritmo, fornecendo como saída os instantes de ocorrência das DOEs emitidas por cada um dos peixes. Esses instantes são o produto final do nosso método, e podem ser analisados e estudados para pesquisar o comportamento e os esquemas de codificação que ocorrem na eletrocomunicação entre peixes.

Em experimentos que não sejam em *closed-loop*, ou seja, quando não for necessário o reconhecimento das DOEs em tempo real, é possível realizar as etapas acima em qualquer ordem. Uma vez gravados em disco, os arquivos de cada etapa devem ser identificados corretamente (como arquivo de **treino** ou arquivo para **discriminação**) quando fornecidos ao software de processamento *off-line*.

Já no caso de experimentos em tempo real, as etapas devem ser conduzidas exatamente na ordem apresentada: o estágio de treino deve ocorrer primeiro para permitir que o algoritmo de classificação supervisionado seja treinado antes que as DOEs possam ser discriminadas no experimento principal.

2.3 Procedimento de validação

Para avaliar se nosso algoritmo de discriminação fornecia resultados precisos, realizamos experimentos nos quais anexamos eletrodos longitudinalmente ao corpo de cada peixe, os quais gravamos além do arranjo de eletrodos fixo que já estava presente no aquário. Os instantes de emissão de DOEs puderam ser obtidos diretamente monitorando quando o sinal proveniente desses novos eletrodos ultrapassava um limiar. Essas medidas diretas foram comparadas, então, à saída de nosso algoritmo de discriminação, para o qual apenas as medidas dos eletrodos fixos foram fornecidas como entradas.

Para manter os eletrodos de medida direta próximos aos peixes, os fios dos mesmos foram entrelaçados a peças de tule de náilon, que por sua vez foram enroladas em volta de cada animal. Dessa forma, o movimento dos peixes foi bastante restringido durante estes testes, exigindo que nós movêssemos manualmente os indivíduos ao redor do aquário, empurrando e puxando os fios, para simular os deslocamentos de posição de um peixe que estivesse nadando

livremente.

Seis pares diferentes de *Gymnotus* sp. foram utilizados para executar este procedimento de validação. Para cada par, dois experimentos foram realizados. No primeiro, ambos os peixes foram restringidos pelo tule de náilon e movidos manualmente ao redor do aquário. Porém, no segundo experimento, um indivíduo arbitrariamente escolhido foi solto, permitindo-se que o mesmo nadasse, enquanto o outro peixe foi mantido preso a seu eletrodo, e portanto teve seus instantes de DOE medidos diretamente.

Capítulo 3

Algoritmo de discriminação

Este capítulo descreve dois algoritmos distintos para discriminação de DOEs de peixes elétricos pulsadores: o método de correlação cruzada (seção 3.1) e o algoritmo proposto por esta tese (seção 3.2). Ambos foram inicialmente implementados visando execução *off-line*, ou seja, assumem que os dados de entrada estão previamente gravados em disco, em vez de processá-los durante o curso do experimento. O conteúdo deste capítulo é baseado em um artigo publicado por nós na revista *Neurocomputing*,[44] e as implementações em software dos algoritmos aqui descritos estão publicamente disponíveis.[45]

O método de correlação cruzada será utilizado somente para fins de comparação de acurácia (seção 5.1), motivo pelo qual será mencionado apenas no presente capítulo e no de resultados. Já o algoritmo proposto por nós tem seu desempenho avaliado na seção 3.3, com o objetivo de identificar os gargalos que dificultam sua implementação de tempo real, e segue sendo utilizado como ponto de partida para o desenvolvimento do sistema de discriminação em tempo real descrito no capítulo 4.

3.1 Trabalho preexistente: correlação cruzada

Antes de discutir o algoritmo proposto por esta tese, começamos com uma breve explicação de como foi implementado o método de correlação cruzada previamente existente na literatura,[4] que utilizaremos para fins de comparação.

Dado um sinal $A[\cdot]$ contendo uma EOD de origem desconhecida, o método calcula (equa-

ção 3.1.1) a correlação cruzada $T_k \star A$ para $k = 1$ e para $k = 2$. Cada valor de k corresponde a um dos peixes da díade, e T_k é o gabarito (*template*) desse peixe.

$$(T_k \star A)[i] = \sum_{j=0}^{l_w - 1} T_k[j] \cdot A[i+j] \tag{3.1.1}$$

Onde l_w é a duração máxima de uma DOE em número de amostras. A seguir, calcula-se um valor $\mathrm{MaxCorr}_k$ (equação 3.1.2) que corresponde à correlação cruzada máxima, em módulo, do k-ésimo peixe, dentre todos os instantes de início i possíveis dentro do sinal $A[\cdot]$.

$$\mathrm{MaxCorr}_k = \max_{i \in [-l_w, l_w]} |(T_k \star A)[i]| \tag{3.1.2}$$

Por fim, caso $(\mathrm{MaxCorr}_1 - \mathrm{MaxCorr}_2) \geq \mathrm{LimiarDecisao}$, o sinal $A[\cdot]$ é classificado como originado do "peixe 1". Caso contrário, é classificado como produzido pelo "peixe 2". O limiar de decisão ($\mathrm{LimiarDecisao}$) não existia no método original. Nós introduzimos esse parâmetro para que pudéssemos variá-lo de forma a plotar as curvas de característica de operação do receptor (COR) apresentadas no capítulo 5. Quando o limiar vale zero, os resultados do método original são recuperados.

Escolhemos os gabaritos T_k por meio de uma busca exaustiva sobre todos os possíveis pares de DOEs extraídos de subconjuntos de 10^3 pulsos de origem conhecida, selecionados aleatoriamente[46] dos dados coletados durante o estágio de treinamento. Normalizamos esses gabaritos de forma que $\max_j |T_k[j]| = 1$. Selecionamos o par de gabaritos que resulte na melhor acurácia de classificação sobre os demais dados coletados no estágio de treino, ou seja, sobre todas as DOEs que não tenham sido selecionadas para fazer parte do subconjunto dentro do qual é realizada a busca exaustiva de gabaritos.

Um grave problema do método de correlação cruzada é a baixa acurácia obtida quando a duração característica dos pulsos de DOE de um dos peixes da díade é muito próxima à do outro peixe. Além disso, esse método não inclui nenhuma técnica para lidar com DOEs sobrepostas. Portanto, nosso maior desafio ao implementar um novo algoritmo foi lidar com esses dois pontos — classificar sinais de acordo com sua estrutura fina, apesar desta variar de acordo com a posição do peixe, e discriminar corretamente as DOEs mesmo nos instantes em

que ambos os peixes disparam quase ao mesmo tempo.

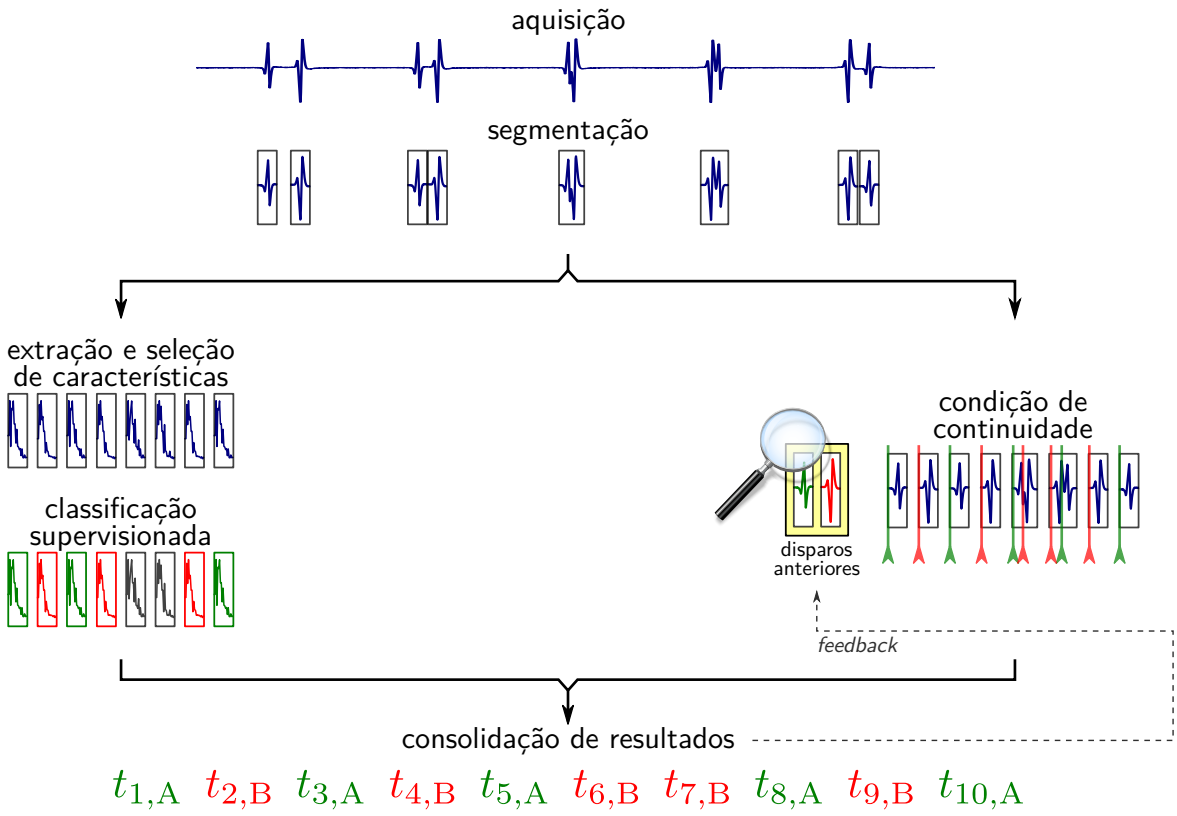

Figura 3.1- Visão geral de nosso algoritmo de discriminação de descargas de órgão elétrico.
Fonte: Elaborada pelo autor.

3.2 Novo estado da arte: algoritmo proposto

A Figura 3.1 apresenta uma visão geral de nosso algoritmo de discriminação de descargas de órgão elétrico. O algoritmo processa segmentos de sinal que podem conter uma ou duas DOEs, respectivamente de um único peixe ou de ambos os peixes de uma díade. A subseção 3.2.1 descreve como esses segmentos são obtidos a partir dos sinais medidos no aquário. Conceitualmente, o algoritmo consiste de duas etapas principais, que ocorrem em paralelo: classificação de segmentos contendo um único peixe (fluxo à esquerda na Figura 3.1) e separação de segmentos contendo DOEs de ambos os peixes (fluxo à direita). O cerne do primeira etapa é uma máquina de vetores de suporte (SVM, do inglês *support vector machine*). A extração e a seleção de características usadas para alimentar a SVM são detalhadas, respectivamente, nas subseções 3.2.2 e 3.2.3, e a aplicação da SVM para classificar os segmentos contendo uma

única DOE é descrita na subseção 3.2.4. A segundo etapa consiste em aplicar uma condição de continuidade para permitir a discriminação de duas DOEs presentes no mesmo segmento, técnica que é reportada na subseção 3.2.5.

3.2.1 Segmentação de DOEs

Tanto os dados provenientes do estágio de treino quanto aqueles coletados no experimento principal são segmentados antes de serem analisados. Esse procedimento tem o propósito de detectar as DOEs, distinguindo-as do ruído de fundo, e de estimar a duração de tempo de atividade das DOEs em cada detecção.

Sinais $A_c[\cdot]$ de todos os canais c passam por um filtro passa-baixas de resposta ao impulso finita (com janela Hamming, suporte 11, e corte em 2500 Hz), diferenciados e elevados ao quadrado. Quando a soma $I[\cdot]$ de todos os sinais resultantes (equação 3.2.3) ultrapassa um limiar t_d, um segmento de atividade do órgão elétrico é detectado.

$$F_c[\cdot] = A_c[\cdot] * \text{Filtro}[\cdot] \tag{3.2.1}$$

$$\Delta F_c[i] = F_c[i+1] - F_c[i] \tag{3.2.2}$$

$$I[i] = \sum_c (\Delta F_c[i])^2 \tag{3.2.3}$$

As fronteiras do segmento (instantes de início e de fim, s e e) são determinadas iterando pelas amostras do sinal somado $I[\cdot]$, em ambas as direções, começando da posição $I[d]$ onde a detecção ($I[d] > t_d$) ocorreu, até que um certo número N_b de amostras contíguas após a fronteira seja encontrado abaixo de um valor mínimo t_b, que é escolhido de forma a ser menor que o limiar de detecção ($t_b < t_d$). Esse procedimento é mostrado na Figura 3.2 e descrito pelas equações 3.2.4 e 3.2.5.

$$\max \quad \{s \in \mathbb{N},\, s < d \,:\, \forall i \in \mathbb{N},\, 1 \leq i \leq N_b \rightarrow I[s-i] < t_b\} \tag{3.2.4}$$

$$\min \quad \{e \in \mathbb{N},\, e > d \,:\, \forall i \in \mathbb{N},\, 1 \leq i \leq N_b \rightarrow I[e+i] < t_b\} \tag{3.2.5}$$

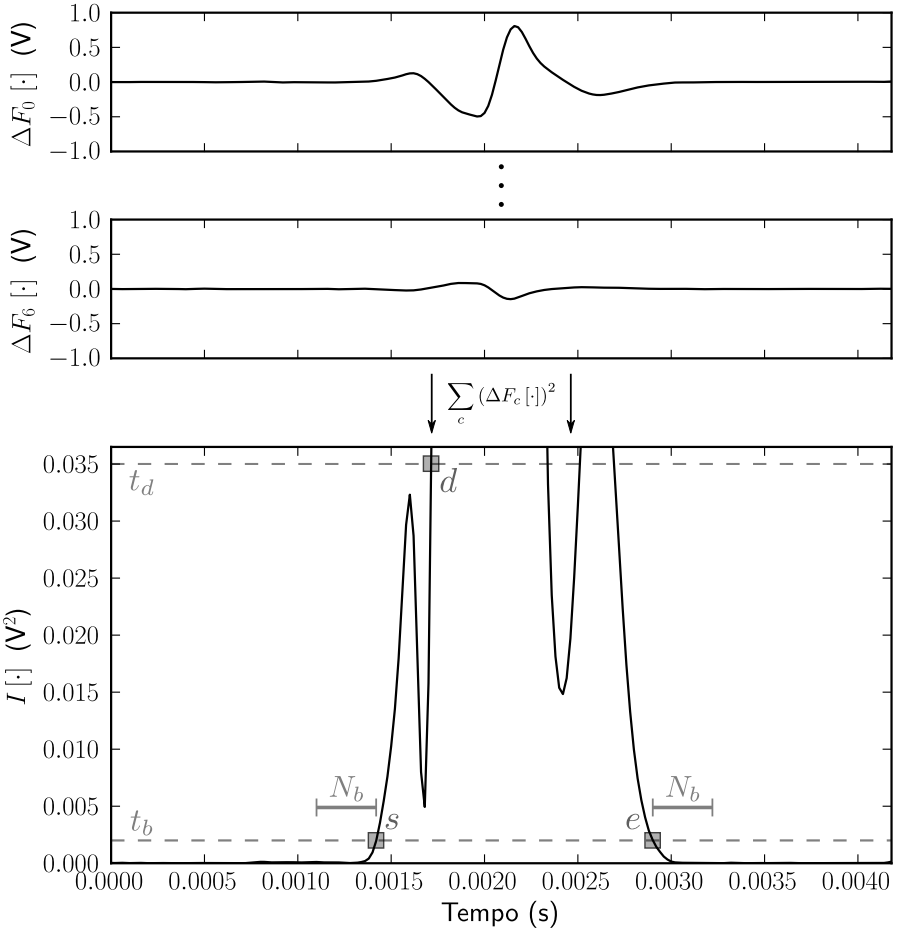

Figura 3.2- Os sinais $\Delta F_c\,[\cdot]$, obtidos filtrando e diferenciando os dados experimentais coletados de cada canal c, são elevados ao quadrado e somados para calcular $I\,[\cdot]$. Quando $I\,[\cdot]$ ultrapassa um limiar t_d, um segmento de atividade de DOE é detectado. Para estabelecer as fronteiras s (e e) do segmento, procuramos por um período "silencioso" de N_b amostras ocorrendo antes (depois) do sinal cair abaixo do limiar t_b.

Fonte: Elaborada pelo autor.

Lembrando o fato de que segmentos de sinal obtidos dos dados de treinamento contém somente DOEs emitidas por um peixe sozinho, centralizamos esses segmentos em uma janela de tamanho padrão l_w, suficiente para acomodar um único pulso de DOE. As fronteiras $s' = {}^{(s+e-l_w)}/_2$ e $e' = s' + l_w$ da janela centralizada podem ser, assim, calculadas a partir das fronteiras s e e do segmento original. Em *Gymnotus* sp., a duração dos pulsos geralmente está na faixa de 1,8 a 2,2 ms, portanto escolhemos um valor $l_w = 128$ (2,56 ms) quando coletamos sinais a 50 kS/s. As janelas centralizadas são usadas para calcular os vetores de características que são empregados para treinar o classificador.

É importante ressaltar que o processo de filtragem e diferenciação do sinal é adotado somente para esta etapa do algoritmo, visando apenas estabilizar a localização de início e fim e a centralização das janelas. Esse sinal transformado é descartado após esta etapa, ou seja, as etapas seguintes do algoritmo utilizam o sinal original obtido do sistema de aquisição.

3.2.2 Extração de características

Um conjunto de características é independentemente calculado para cada canal c do sinal original $A_c[\cdot]$, assim como este era antes de ser filtrado e diferenciado. Nós selecionamos mecanismos de extração de características que produzissem uma saída menos sensível à posição do peixe que os sinais no domínio do tempo. Uma abordagem simples é calcular a transformada de Fourier $\tilde{A}_c[\cdot] = \mathfrak{F}\{A_c[s' \cdots e']\}$ das janelas de sinal dos pulsos, tomar o módulo $\left|\tilde{A}_c[\cdot]\right|$ dos valores complexos resultantes — que representa a amplitude de cada componente de frequência sem a informação de fase — e finalmente normalizar o vetor, dividindo todos os valores pelo maior ($\left|\tilde{A}_c[\cdot]\right| / \max_i \left|\tilde{A}_c[i]\right|$).

Todavia, existe o relato de um experimento[47] no qual algumas espécies de peixe eram capazes de distinguir entre pares de pulsos de DOE artificiais que possuíam as mesmas amplitudes no espectro de frequências. No trabalho citado, as DOEs artificiais eram construídas superpondo duas componentes deslocadas no tempo. Desta forma, as DOEs diferiam somente na fase do espectro de frequências, que não seria distinguida olhando para $\left|\tilde{A}_c[\cdot]\right|$. Formulamos, então, a hipótese de que, se os peixes são capazes de sentir essas variações em DOEs artificiais, então é provável que descargas reais contenham, pelo menos em alguns casos, componentes

localizadas no tempo, análogas às componentes deslocadas no tempo do sinal artificial. Por exemplo, alguma faixa de frequências pode ser particularmente mais predominante no começo das DOEs emitidas por um determinado indivíduo. Esse tipo de informação é de grande utilidade para o classificador, e pode ser extraída por meio de técnicas de tempo-frequência, tais como as transformadas *wavelet*.

Transformadas de pacote *wavelet* formam árvores de decomposição que, a cada nível, refinam a localização do sinal na frequência, ao mesmo tempo em que perdem a localização no tempo. Nós escolhemos a transformada complexa de dupla árvore de pacote *wavelet* (DT-CWPT, do inglês *dual-tree complex wavelet packet transform*) por esta ser praticamente invariante a deslocamentos no tempo, apesar de ser capaz de percebê-los quando o nível de decomposição permite localização temporal suficiente. A transformada de pacote *wavelet* comum (DWPT, do inglês *discrete wavelet packet transform*) não possui essa propriedade, sendo assim suscetível a artefatos induzidos por variação de fase em **todos** os níveis de decomposição. Esse efeito é inconveniente, pois o movimento dos peixes causa mudanças de fase nos sinais de DOE que não desejamos levar em conta.

Nós implementamos a DT-CWPT usando filtros Q-shift[48] de suporte 20, e Daubechies com sete momentos de anulação. As árvores duplas são combinadas para obter uma única árvore, composta pelo valor absoluto de cada componente. Como a transformada é calculada em janelas de $l_w = 128$ amostras, as árvores decompõem os sinais em $\log_2(l_w) = 7$ níveis, cada um contendo 128 componentes. Cada nível de uma árvore é normalizado de forma independente antes de ter suas componentes adicionadas aos vetores de características.

Em resumo, a transformada de Fourier e a DT-CWPT fornecem, respectivamente, $l_w/2 = 64$ e $l_w \cdot \log_2 l_w = 896$ componentes que, uma vez normalizadas, podem fazer parte do vetor de características.

3.2.3 Seleção de características

O número de componentes fornecidas pelas transformadas de Fourier e DT-CWPT é grande (960 no total). Se todas essas componentes fossem usadas para compor o vetor de características, os tempos de treinamento seriam excessivamente longos, e o modelo provavelmente

sofreria problemas de sobreajustamento (*overfitting*). Portanto, selecionamos somente 20 dessas características para alimentar o classificador.

Como primeiro passo para o processo de seleção de características, para cada componente do sinal, construímos dois histogramas, um para cada peixe, como ilustrado na Figura 3.3. Cada barra abrange o mesmo intervalo da abscissa em ambos os histogramas, portanto a área sobreposta, destacada na figura, pode ser facilmente calculada integrando sobre a altura da menor barra em cada ponto.

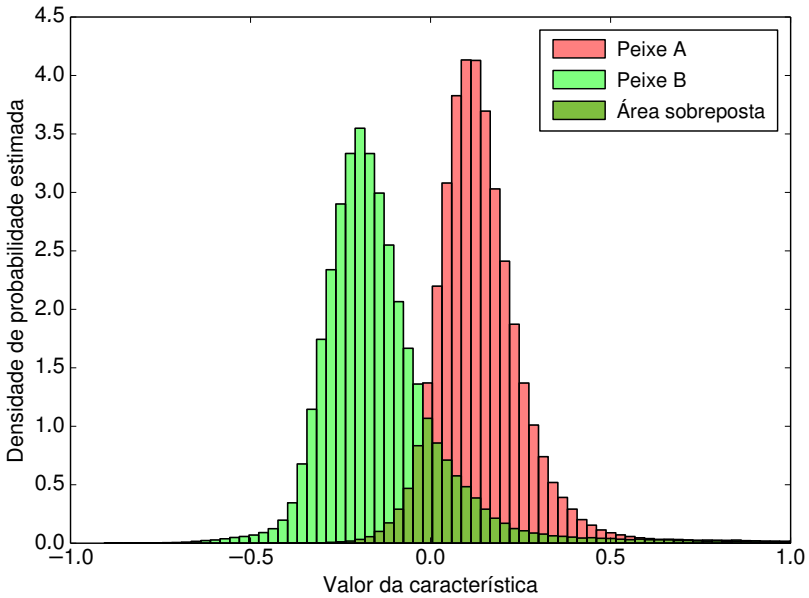

Figura 3.3- O histograma de uma componente Fourier ou *wavelet* pode ser usado para estimar a distribuição de probabilidade dos valores dessa componente. A área sobreposta entre os histogramas da componente em peixes distintos é interpretada como uma medida da incerteza de classificação caso apenas essa característica do sinal fosse conhecida. Nossa abordagem é escolher as componentes que apresentem a menor área sobreposta em seus histogramas.

Fonte: Elaborada pelo autor.

A menor área sobreposta possível (zero) seria obtida em uma situação ideal na qual as DOEs do conjunto de treinamento pudessem ser perfeitamente classificadas usando somente essa característica. A área máxima (um) seria atingida no pior caso, quando a distribuição de probabilidade da componente for praticamente a mesma em ambos os peixes, o que significaria que a componente é inútil para classificação se usada sozinha.

Mesmo que as componentes do sinal não possam ser consideradas independentes umas das outras, assumimos por simplicidade que um bom conjunto de características pode ser construído selecionando aquelas que apresentem a menor área sobreposta em seus histogramas.

Para este problema, verificamos empiricamente que essa forma simples de abordagem de filtro[49] funciona bem para a seleção de características.

3.2.4 Classificação supervisionada

Adotamos classificadores SVM pois estes frequentemente produzem bons resultados para uma grande variedade de problemas.[50] Além disso, bibliotecas de software maduras e otimizadas implementando os algoritmos de treinamento de SVM, como a `libsvm`,[51] estão prontamente disponíveis. Cada característica fornecida ao classificador é redimensionada para o intervalo $[-1, 1]$, de forma a evitar que características abrangendo as maiores faixas de valores numéricos tornem-se dominantes. Nossa implementação usa por padrão o núcleo de função de base radial (RBF, do inglês *radial basis function*), que geralmente é considerado uma boa primeira escolha, já que dependendo dos parâmetros de treinamento, ele pode comportar-se também como os núcleos lineares ou sigmoides.[52]

Sinais de DOE bem-formados (que definimos como aqueles que não estejam saturados e que possuam um bom SNR, ao avaliarmos a amplitude da DOE) obtidos de um peixe sozinho durante o estágio de treino são distribuídos aleatoriamente[46] em três conjuntos, cada um contendo cerca de 10^4 DOEs quando os dados de treino são capturados por 15 minutos. Um deles, denominado conjunto de treinamento, é usado para treinar de fato o modelo SVM; outro, chamado conjunto de validação, para contabilizar o número de erros em uma busca com objetivo de encontrar os melhores parâmetros para o treinamento com *soft-margin* e para o núcleo RBF (C e γ); e o terceiro, denominado conjunto de teste, para estimar o desempenho final da SVM na classificação de descargas provenientes de um único peixe.

Uma vez que o modelo SVM esteja treinado, dados coletados de um experimento com uma díade podem ser processados. Segmentos de sinais obtidos desses dados podem conter uma DOE emitida por um único peixe ou DOEs disparadas por ambos os peixes quase ao mesmo tempo. Diversos critérios são verificados para estabelecer com uma baixa taxa de falsos negativos (alta especificidade) se um certo segmento contém uma única DOE.

Primeiramente, desconsideramos segmentos cujo comprimento $(e - s)$ seja maior que $\max\left\{\bar{l_1} + \sigma_{l_1}, \bar{l_2} + \sigma_{l_2}\right\}$, onde $\bar{l_j}$ é a média e σ_{l_j} é o desvio padrão do comprimento dos

segmentos de sinal presentes nos dados de treinamento coletados do j-ésimo peixe sozinho. Nós também rejeitamos segmentos em que menos que N_r dos canais disponíveis tenham capturado sinais bem-formados (bom SNR, não-saturados), e só consideramos pares de segmentos adjacentes se o intervalo de tempo entre os instantes de início de cada segmento for menor que o período mínimo de descarga que pode ser atingido por um único peixe, caso no qual as DOEs foram provavelmente emitidas por indivíduos diferentes.

Finalmente, o modelo SVM é empregado para calcular[53,54] os estimadores de Platt p_c de cada canal c que contenha um sinal bem-formado dentro de cada segmento. Se um dos produtos $\prod_c p_c$ ou $\prod_c (1 - p_c)$ estiver acima de um certo limiar t_p, o segmento associado é marcado como sendo emitido pelo primeiro ou pelo segundo peixe, respectivamente. Segmentos pertencendo a um par adjacente (segundo os critérios acima) devem conter marcas correspondentes a peixes diferentes, caso contrário a classificação é considerada incorreta.

Geralmente adotamos $N_r = 2$ e $t_p = 0{,}95$. Quanto maiores os valores de N_r e de t_p, maior a especificidade total do classificador de peixes disparando sozinhos. Não recomendamos escolher $N_r > 2$ porque é raro observar uma DOE bem-formada em mais de dois canais ao mesmo tempo. Escolher $N_r = 1$ pode ser efetivo se um peixe houver permanecido por vários segundos do experimento em uma posição do aquário na qual uma DOE bem-formada for capturada por um único canal, mas nesses casos recomendamos compensar a especificidade aumentando o valor de t_p para valores próximos a $0{,}99$. Erros de classificação nesta etapa do algoritmo propagarão para o próximo passo (subseção 3.2.5), portanto forçar uma alta especificidade é importante para obter um erro final de discriminação na ordem de partes por milhão, apesar do erro de classificação de um único canal atingir a faixa das centenas de partes por milhão.

3.2.5 Condição de continuidade

Sempre que um par adjacente de segmentos satisfazendo os critérios de alta especificidade é detectado, para ambos $j \in \{1, 2\}$, vetores $L_{j,c}[\cdot]$ são inicializados com sinais de todos os canais c, sejam eles bem-formados ou não, pertencentes ao segmento classificado como contendo uma única DOE, emitida pelo j-ésimo peixe. Então, uma condição de continuidade é imposta para

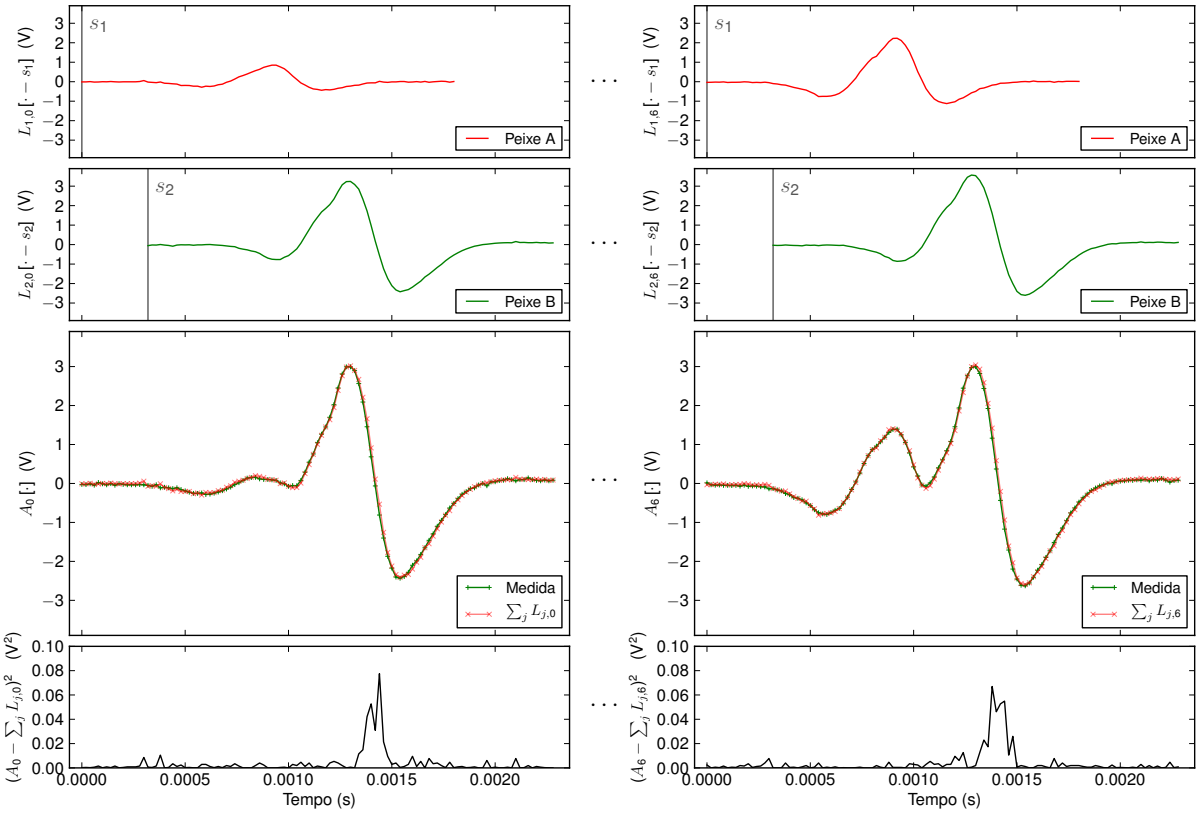

Figura 3.4- Um segmento do sinal $A_c[\cdot]$ pode conter DOEs emitidas por ambos os peixes quase ao mesmo tempo. Para identificar a posição s_j onde a DOE emitida pelo j-ésimo peixe começa dentro do segmento, nós exploramos a propriedade de continuidade das formas de onda de DOE emitidas por cada peixe. A figura é organizada como uma tabela, onde cada coluna contém um gráfico apresentando dados relacionados a um certo canal c do sinal. As primeiras duas linhas mostram os vetores $L_{j,c}$, que contém a última DOE previamente reconhecida como sendo emitida pelo j-ésimo peixe, deslocada para a posição s_j sendo avaliada. Na terceira linha, ambos os vetores deslocados são somados, e os $\sum_j L_{j,c}$ resultantes são comparados aos sinais medidos (A_c). Neste exemplo, não ocorre saturação, portanto não há necessidade de se aplicar a função Ξ definida no texto. Os instantes s_j são escolhidos para minimizar a soma de todas as distâncias ao quadrado, mostradas na quarta linha da figura.

Fonte: Elaborada pelo autor.

permitir o reconhecimento das DOEs presentes nos segmentos de sinal subsequentes, até que o próximo par adjacente obedecendo aos critérios seja encontrado. Nós assumimos que as formas de onda das DOEs produzidas por um indivíduo variam de forma suave, porque a posição de um peixe no aquário é uma função contínua do tempo. Essa hipótese é similar à utilizada por uma abordagem já estabelecida[12] na literatura sobre discriminação de sinais neurais (*spike sorting*) para resolver o problema do movimento de deriva de eletrodos.

A dissociação de segmentos de sinais contendo DOEs de ambos os peixes é, portanto, formulada como um problema de minimização de distância euclidiana. Os instantes de início s_j das DOEs emitidas pelo j-ésimo peixe são encontradas deslocando os vetores $L_{j,c}[\cdot]$ para as posições s_j sendo testadas, somando os vetores deslocados correspondentes a peixes diferentes, e comparando o sinal resultante ao segmento atual $A_c[s \cdots e]$, assim como definido pela equação 3.2.6 e ilustrado na Figura 3.4.

$$\underset{s_1, s_2}{\operatorname{argmin}} \sum_{i,c} \left(A_c[i] - \Xi\left(L_{1,c}[i - s_1] + L_{2,c}[i - s_2]\right)\right)^2 \tag{3.2.6}$$

Em nossa notação, $L_{j,c}[i]$ é tomado como zero sempre que i estiver fora dos limites ($0 \leq i < l_w$). A função $\Xi(V[\cdot])$ modela o efeito de saturação da saída dos amplificadores. Se a tensão máxima de saída for dada por v_{sh} e a mínima por v_{sl}, então Ξ pode ser definido pela equação 3.2.7.

$$\Xi(V[k]) = \begin{cases} v_{sh}, & \text{se } V[k] \geq v_{sh} \\ v_{sl}, & \text{se } V[k] \leq v_{sl} \\ V[k], & \text{caso contrário} \end{cases} \tag{3.2.7}$$

Quando testamos diferentes valores s_j, permitimos que s_1 assuma um único valor tal que $L_{1,c}[i - s_1]$ seja sempre tomado como zero, ou seja, fazendo com que $L_{2,c}[\cdot]$ sozinho seja comparado a $A_c[s \cdots e]$, e vice-versa. Desta forma, o algoritmo de minimização da distância euclidiana pode ser empregado também para detectar segmentos de sinais contendo uma única DOE que foram descartados pelos critérios previamente aplicados, devido à sua alta especificidade. Quando esses segmentos são identificados, o algoritmo atualiza os vetores

$L_{j,c}[\cdot]$ correspondentes ao peixe reconhecido j, de forma que estes reflitam as formas de onda mais recentes.

3.3 Identificação de gargalos de desempenho

A implementação *off-line* do algoritmo de discriminação[45] foi realizada em linguagem C++ paralelizada com OpenMP.[55] A fim de aferir quais etapas do algoritmo necessitavam mais urgentemente ser implementadas em hardware ou otimizadas em software (por exemplo, reescrevendo parte do código com o uso de instruções vetoriais) para que a discriminação pudesse atender a requisitos de tempo real, avaliamos o desempenho da implementação *off-line* quando executada em um computador com processador Intel Core i3-2120 de 3,30 GHz.

Medimos independentemente o tempo médio de execução de diferentes etapas do algoritmo, como exibido na Figura 3.5, passando como entrada para o software um conjunto de dados medido em um aquário com 7 canais de saída. O tempo de execução para um aquário com mais canais (por exemplo o aquário de 11 canais descrito na seção 2.1) seria maior, proporcionalmente ao número de canais.

A aplicação do método de imposição da condição de continuidade, por meio da resolução da equação 3.2.6, é realizada na etapa denominada "sobreposição de disparos anteriores", que claramente domina o tempo de execução. Entretanto, nossa implementação dessa etapa utiliza janelas de tamanho variável para reduzir, quando possível, o comprimento dos vetores envolvidos, com o objetivo de melhorar o tempo total de execução para processar uma série completa de dados. Isso significa que o tempo médio não é representativo do pior caso, que seria a medida mais importante em uma implementação de tempo real.

Para permitir a visualização da variabilidade e do pior caso do tempo de execução dessa etapa, apresentamos um histograma desse tempo na Figura 3.6. Pode-se observar que, de fato, há uma grande variabilidade do tempo de execução. No pior caso, o tempo de execução ($\approx$ 14,5 ms) vale mais que o dobro do tempo médio ($\approx$ 6,5 ms) da etapa.

Apesar de ser necessária uma intervenção nesse gargalo para permitir que o método seja utilizado em tempo real, o desempenho não é de todo insatisfatório considerando-se que se trata de uma implementação *off-line*. Somando o tempo médio de execução de todas as

etapas apresentadas na Figura 3.5, obtemos o total de 7,13 ms, necessários para lidar com cada segmento de atividade de DOE detectado na entrada. Isso significa que, com a implementação em software, é possível processar cerca de 140 segmentos por segundo. Uma vez que a taxa de disparo média de um *Gymnotus* sp. geralmente permanece abaixo dos 70 Hz, o desempenho da implementação *off-line* chega próximo ao necessário para uma implementação *on-line* para séries de dados gravadas com 7 canais, ou seja, é capaz de discriminar a série em tempos da mesma ordem que a duração do experimento.

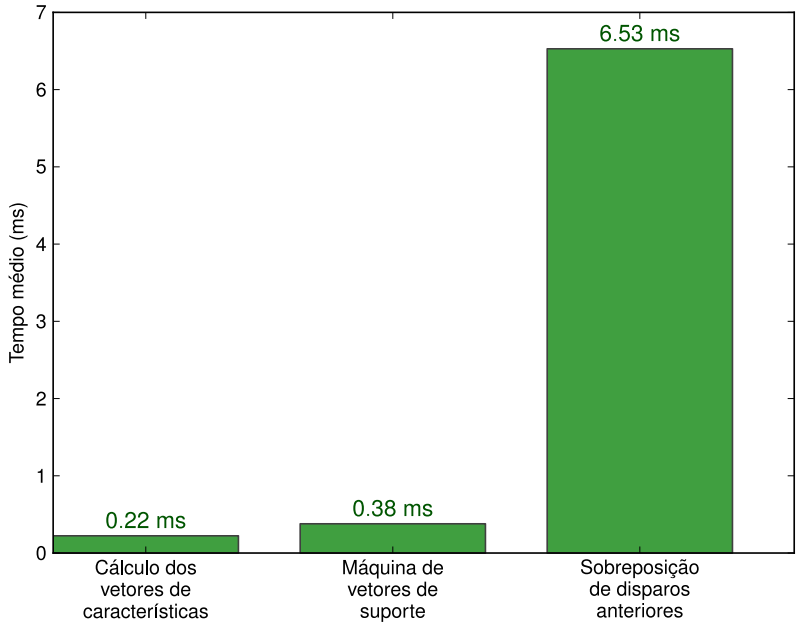

Figura 3.5- Tempo médio de execução de cada etapa do algoritmo de discriminação implementado em software. A etapa de "cálculo dos vetores de características" compreende a obtenção e a normalização das transformadas de Fourier e DT-CWPT de todos os canais de uma janela de sinal. A etapa de "máquina de vetores de suporte" corresponde à execução do classificador SVM. A etapa de "sobreposição de disparos anteriores", que domina o tempo de execução, refere-se à aplicação do método de imposição da condição de continuidade.

Fonte: Elaborada pelo autor.

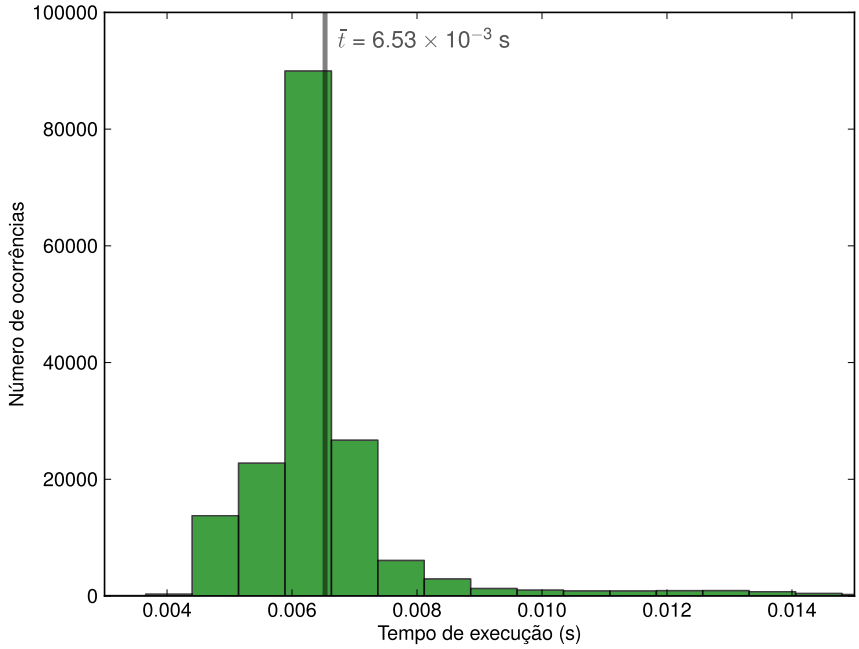

Figura 3.6- Histograma do tempo de execução da etapa de "sobreposição de disparos anteriores". Além de ser a etapa do software que consome o maior tempo de execução, ocorre uma grande variabilidade desse tempo, prejudicando a utilização desse módulo do software em um sistema capaz de gerar resultados em tempo real.

Fonte: Elaborada pelo autor.

Capítulo 4

Implementação de tempo real

Uma vez desenvolvida e validada a implementação *off-line* de nosso algoritmo, que foi realizada totalmente em software, passamos a abordar o problema de transformá-la em uma implementação de tempo real. Adotamos uma metodologia de co-projeto de hardware/software,[31] na qual priorizamos a implementação em hardware das etapas com maior gargalo de desempenho ou de latência, com o objetivo de atender aos requisitos desejados.

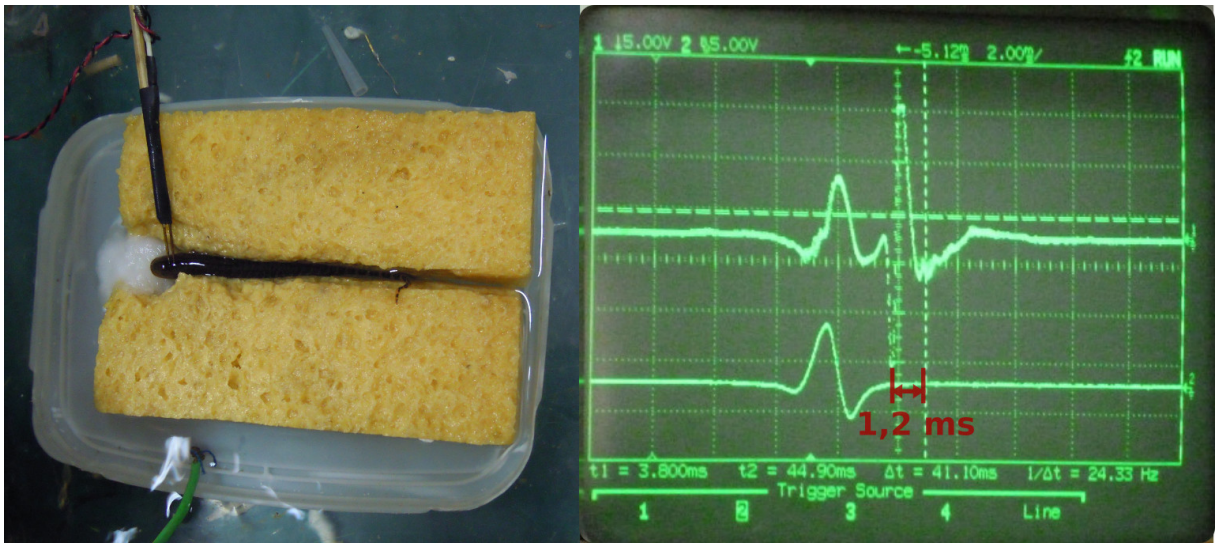

Figura 4.1- Medida intracraniana do núcleo magnocelular do cérebro do peixe. À esquerda, montagem experimental (Instituto de Investigaciones Biológicas Clemente Estable, Uruguai). À direita, tela do osciloscópio mostrando: no canal 1, sinal coletado do núcleo magnocelular; no canal 2, EOD coletada na água. Existe um atraso de cerca de 1,2 ms entre o término da EOD e o sinal ser observado no núcleo magnocelular.

Fonte: Fotografias cedidas por RAFAEL TUMA GUARIENTO (2015).

A latência do sistema sensorial elétrico de *Gymnotus* sp. é da ordem de 1 a 2 ms, estimada com base no tempo típico para um potencial de ação ser observado no núcleo magnocelular

do cérebro de um peixe após o término de sua DOE.[32] A Figura 4.1 afere esse tempo por meio de uma medida intracraniana, comparando-a à DOE observada na água. É desejável que nosso discriminador possua, no máximo, a mesma latência do sistema sensorial natural. Isso permitiria, por exemplo, que ao simularmos um peixe virtual tendo como base a saída do discriminador, o modelo tivesse acesso às mesmas informações que os peixes reais em estudo.

De acordo com a Figura 3.5, o maior empecilho para alcançar uma latência máxima em torno de 1 ms é a etapa de sobreposição de disparos. Mesmo paralelizando e otimizando essa etapa do software com todos os métodos disponíveis, não é possível aproximar-se do requisito de tempo. A solução para explorar todo o paralelismo existente, permitindo executar essa etapa em tempo hábil, é construir uma arquitetura de hardware dedicada a essa tarefa.

As demais etapas da implementação *off-line*, somadas, executam em 0,60 ms quando processam dados capturados de um aquário de 7 canais, ou 0,95 ms para um aquário de 11 canais. É possível manter essas etapas implementadas em software, executar a etapa de sobreposição de disparos em segundo plano no hardware dedicado, e consolidar o resultado de discriminação quando tanto o hardware como o software houverem concluído suas tarefas. Entretanto, os valores mencionados são apenas uma média do tempo de execução. O tempo pode sofrer variações devido a fatores como presença ou ausência das informações a serem processadas no *cache* do processador. No caso de algoritmos que executem em múltiplas *threads*, como a etapa de SVM da implementação *off-line*, é necessário ter ainda mais cuidado com esta questão, pois surgem outras fontes de variabilidade do tempo, devido à comunicação entre diferentes núcleos do processador. Portanto, algumas otimizações ainda são necessárias para garantirmos um máximo de 1 ms no pior caso.

Além disso, as rotinas de software precisam ser adaptadas para executar em um sistema operacional de tempo real (RTOS), de forma a garantir que serão tratadas com a devida prioridade. Em um sistema operacional de propósito geral, estas poderiam ser pausadas a qualquer momento por um intervalo de tempo não determinístico, causando grande variabilidade (*jitter*) do tempo de execução.

O RTOS escolhido foi o RTAI,[34] que possui a vantagem de executar lado a lado com o Linux no mesmo processador. Desta forma, os módulos em software do discriminador e outras

tarefas que necessitem de determinismo podem ser implementadas no domínio de tempo real do RTAI, ao mesmo tempo que o rico ecossistema do Linux permanece disponível para a elaboração de interfaces gráficas e ferramentas em geral para a operação do sistema.

Os módulos de hardware deste projeto foram implementados em Bluespec,[35] uma linguagem de descrição de hardware de alto nível com tipagem forte e paradigma funcional. Apesar de ser uma linguagem de alto nível, Bluespec oferece total controle sobre a microarquitetura do hardware, propiciando ao projetista previsibilidade do desempenho e da área ocupada pela lógica ao longo de seu desenvolvimento.[56]

Um projeto em Bluespec é organizado em módulos e regras. Os módulos fornecem interfaces organizadas como um conjunto de métodos que podem ser chamados para acessar ou modificar o estado interno do módulo. Operações que causem mudanças de estado (efeitos colaterais, na terminologia de programação funcional) são claramente separadas de operações somente-leitura com o auxílio de uma mônada[57] chamada `Action`, e por isso toda expressão que modifique estado possui um tipo encapsulado pelo tipo `Action`. Módulos podem ser elaborados estaticamente diversas vezes, permitindo representar estruturas de circuito complexas.

Regras são entidades que descrevem as conexões entre módulos e, fundamentalmente, são elas que definem a dinâmica do hardware. Elas são formadas por um conjunto de ações e por um predicado booliano (guarda), que define uma condição explícita necessária para permitir a execução das ações (disparo da regra). Durante um único ciclo de relógio (*clock*), é garantido que uma regra execute completamente ou que simplesmente não dispare, propriedade que é conhecida como atomicidade. O disparo da regra também pode ser afetado por condições implícitas, que podem ser atribuídas a qualquer expressão em Bluespec. O compilador propaga condições implícitas até o predicado da regra que de fato executa a ação ou consulta o valor da expressão correspondente. Condições implícitas são geralmente atribuídas a métodos, servindo como uma maneira efetiva de especificar contratos entre módulos. O compilador define uma ordem de execução para as regras, permitindo que um escalonador em hardware seja gerado de acordo com o formalismo de sistemas de redução.[58]

O código fonte completo do projeto de hardware e de software de tempo real descrito neste capítulo está publicamente disponível.[59] Iniciamos a discussão na seção 4.1, com uma visão

geral de toda a arquitetura de hardware desenvolvida, apresentando os submódulos existentes logo abaixo do módulo principal do projeto. Na seção 4.2, mostramos como esse módulo é configurado para comunicar-se com o barramento PCI Express do computador. Em seguida, as seções 4.3, 4.4, 4.5 e 4.6 detalham a implementação de alguns dos principais submódulos do projeto. A seção 4.7, última deste capítulo a discutir temas relacionados a hardware, expõe um trabalho[60] decorrente deste projeto — um conjunto de ferramentas que facilita o uso do barramento JTAG[61] para teste e depuração de hardware — e apresenta dois breves exemplos de uso dessas ferramentas. As duas últimas seções tratam de temas relacionados a software de tempo real. A seção 4.8 aborda as otimizações realizadas nas etapas do algoritmo que foram mantidas em software. Por fim, a seção 4.9 apresenta as adaptações realizadas no RTAI para que este possa executar software otimizado com o conjunto de instruções Intel AVX.[33]

4.1 Visão geral da arquitetura

A Figura 4.2 apresenta uma visão geral da arquitetura do módulo principal de hardware desenvolvido como parte desta tese. Esse módulo comunica-se com o mundo exterior por meio das seguintes interfaces:

- **AvalonSlave**, **AvalonMaster** e **InterruptSender**: Implementam interfaces compatíveis com o padrão de barramento Avalon, da Altera. Na seção 4.2, explicamos como essas interfaces Avalon são conectadas ao barramento PCI Express.

- **ADC**: Conversor analógico-digital (AD) responsável pela leitura dos sinais de entrada provenientes do aquário. A subseção 4.3.2 detalha como o conversor comunica-se com a lógica implementada no FPGA.

- **DigitalOut**: Saídas digitais que podem ser usadas para medir a latência do sistema com o auxílio de um osciloscópio (seção 5.4), ou para controlar a geração de estímulos por um hardware externo (seção 6.1).

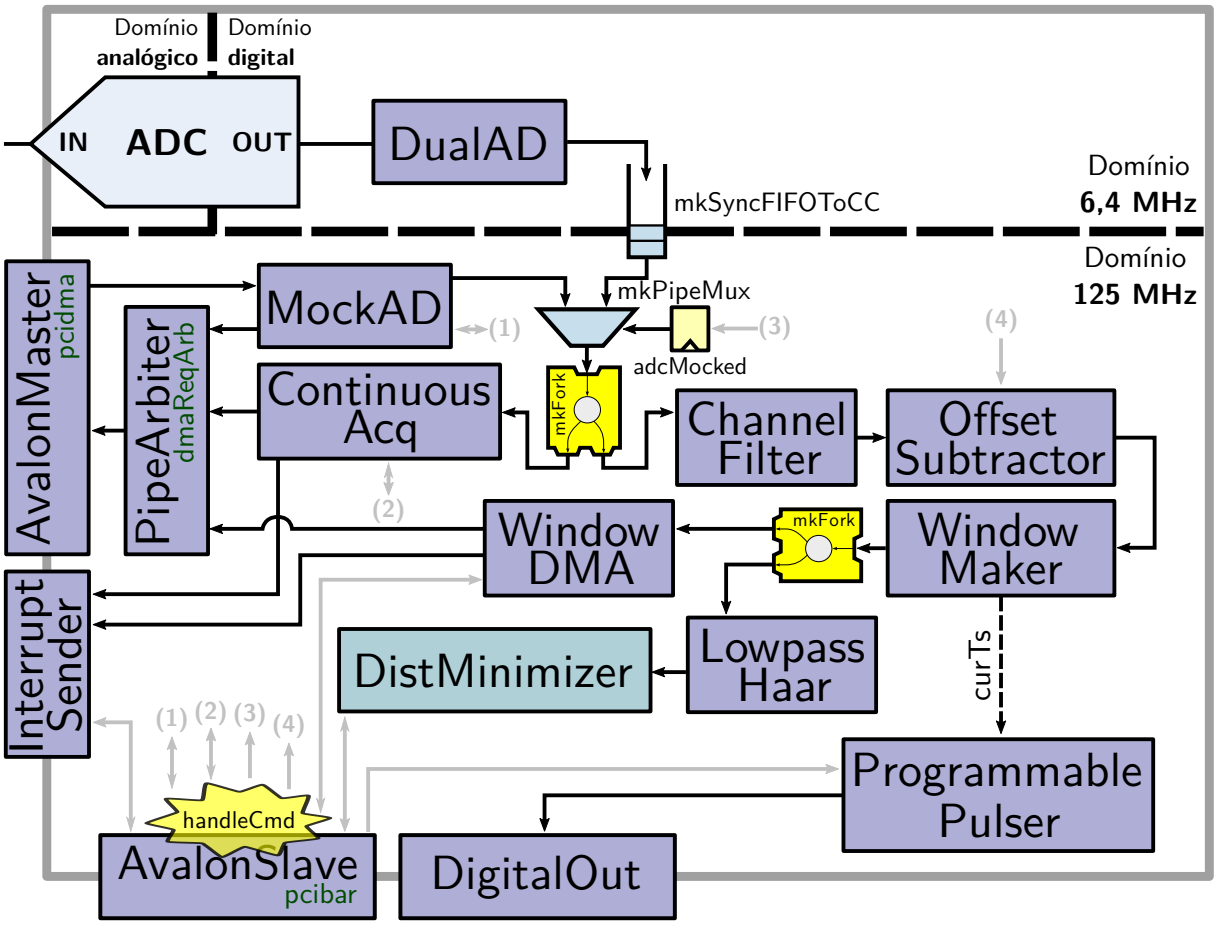

Figura 4.2- Visão geral do módulo de hardware principal. Os retângulos coloridos dentro do quadro cinza representam submódulos, e as setas indicam o fluxo de dados entre os diferentes módulos. Os retângulos posicionados sobre as bordas do quadro são as interfaces do módulo principal com o mundo exterior. As setas mais claras são os caminhos pelos quais circulam os comandos disponibilizados ao software de tempo real para controlar o hardware. Esses comandos são gerenciados pela regra `handleCmd`, com base em requisições recebidas pela interface `AvalonSlave`.

Fonte: Elaborada pelo autor.

Dentro do quadro cinza da figura, mostramos os diversos submódulos que são instanciados logo abaixo do módulo principal:

- **DualAD**: Módulo que controla dois conversores AD de 8 canais cada um. É descrito em maiores detalhes na subseção 4.3.2. Esse módulo fornece como saída tuplas contendo o número de um canal (de 0 a 15) e o valor da amostra adquirida desse canal. Demos a esse tipo de tupla o nome `ChSample`.

- **ChannelFilter**: Apesar de suportarmos 16 canais de entrada, nem sempre todos esses canais estão conectados a sinais do aquário. Em particular, os aquários disponíveis em nosso laboratório fornecem apenas 7 ou 11 canais de saída. Este módulo tem o propósito

de descartar amostras provenientes de canais que não estejam habilitados em uma lista fornecida durante a compilação (`enabledChannels`).

- **OffsetSubtractor**: O conversor MAX1280, utilizado por nosso sistema de aquisição customizado, suporta apenas um modo bipolar emulado, no qual o centro da faixa mensurável (1,25 V) é subtraído digitalmente pelo próprio conversor. Por isso, optamos por utilizar o modo unipolar do conversor e deixar a subtração a cargo de nossa própria lógica. Este módulo permite que um valor diferente seja configurado para ser subtraído de cada canal, corrigindo quaisquer *offsets* existentes no sinal.

- **WindowMaker**: Módulo delimitador de janelas, detalhado na seção 4.4. Recebe como entrada uma tupla `ChSample` e fornece como saída um *tagged union* (tipo algébrico) que pode conter um `ChSample` ou um marcador de fim de janela (`EndMarker`) acompanhado de uma estrutura com informações da janela (`WindowInfo`). Além dessa saída, este módulo também expõe em sua interface o valor do instante de tempo atual (`curTs`), de acordo com o número de amostras que foram processadas até o momento.

- **DistMinimizer**: Módulo de sobreposição de disparos, que testa todas as posições possíveis de disparos anteriores dentro da janela atual, escolhendo a melhor pelo critério de mínima distância total. A implementação deste módulo é discutida na seção 4.6.

- **LowpassHaar**: Filtro decimador (passa-baixas Haar). À taxa de amostragem de 50 kS/s por canal, janelas contendo disparos sobrepostos de ambos os peixes podem ter comprimento de até 256 amostras. Quando `DistMinimizer` é implementado para receber janelas desse comprimento, o FPGA adotado neste trabalho possui blocos lógicos o suficiente para habilitar no máximo 8 canais (subseção 5.3). Inserindo `LowpassHaar`, a taxa de amostragem é reduzida para 25 kS/s, permitindo que sejam habilitados todos os 16 canais disponíveis, com a desvantagem de reduzir de 20 μs para 40 μs a resolução temporal dos *timestamps* de DOEs detectadas em janelas que apresentem sobreposição.

- **WindowDMA**: Módulo de transferência de janelas, descrito na seção 4.5. Recebe como entrada o tipo algébrico produzido pelo módulo `WindowMaker` e fornece como saída um

fluxo de requisições de escrita na memória RAM do computador. Conecta-se também à interface que envia interrupções para o software de tempo real.

- **MockAD** e **ContinuousAcq**: Pertencem à infraestrutura de teste do sistema de aquisição (subseção 4.3.3). Quando ativado por meio do registrador `adcMocked`, o módulo `MockAD` substitui os sinais provenientes do conversor AD por sinais lidos da memória do computador. Já o módulo `ContinuousAcq` pode ser utilizado para transferir amostras de todos os 16 canais de forma contínua para o computador, como se o sistema fosse uma placa de aquisição comercial simples.

- **PipeArbiter**: Arbitrador dinâmico *round-robin* baseado no módulo `mkArbiter` da biblioteca padrão do Bluespec. Permite que diferentes módulos de nosso sistema enviem requisições à interface `AvalonMaster`.

- **ProgrammablePulser**: Gerador de pulsos programável. Recebe comandos do software para gerar um pulso em uma das saídas digitais do sistema quando o *timestamp* atual (`curTs`) possuir determinado valor.

Como nossa arquitetura processa um sinal digital em estágios, formando uma estrutura de *pipeline*, decidimos padronizar a maioria das linhas de comunicação entre módulos (representadas por setas na figura) adotando a interface `PipeOut` definida pela biblioteca PAClib (*Pipelined Architecture Composers library*), que acompanha a distribuição padrão do Bluespec.

A PAClib fornece um conjunto de blocos de construção plugáveis baseados na interface `PipeOut`, que podem ser parametrizados pelo projetista com funções de transformação de dados, estruturas, *buffers* e diferentes tipos de dados. Um desses blocos é o `mkFork`, representado por uma peça amarela (retangular e com encaixes) na figura, que recebe uma interface `PipeOut` como entrada, e fornece duas interfaces de mesmo tipo na saída, replicando os dados de entrada em ambas as saídas.

A interface `PipeOut` é composta por métodos que mimetizam a interface de saída de uma fila do tipo FIFO (*first-in first-out*):

- `notEmpty`: retorna um valor booliano indicando se existe alguma saída pronta para ser retirada do módulo;

- `first`: consulta o valor da próxima saída sem retirá-la do módulo;
- `deq`: sinaliza ao módulo que um elemento foi retirado da saída.

O único método capaz de causar alterações em elementos de estado do circuito (registradores e memórias) é o `deq`, que possui, portanto, o tipo `Action`. Todo módulo que implementa essa interface define como condição implícita para os métodos `first` e `deq` o mesmo booliano retornado por `notEmpty`, convenção que facilita muito o projeto de *pipelines* elásticos e de lógica insensível à latência.[62–64]

Uma consequência interessante de nossa lógica ser insensível à latência é que esta não precisa necessariamente receber dados de entrada a uma taxa de amostragem fixa, recurso que é explorado para permitir que o mesmo hardware seja utilizado para experimentos em tempo real e para acelerar o processamento *off-line* de dados previamente adquiridos, discussão que será retomada na subseção 4.3.3.

4.2 Infraestrutura de comunicação via PCI Express

Apesar de suportar diretamente os barramentos AXI, APB e AHB, utilizados em sistemas com processadores ARM e outras arquiteturas embarcadas, a ferramenta de integração de sistemas da Altera (Qsys) suporta o barramento PCI Express apenas por meio de uma conversão para o padrão de barramento da Altera, denominado Avalon. Como mostra a Figura 4.3, o módulo `pcie_hard_ip` (*IP Compiler for PCI Express*) é responsável por essa conversão. As seguintes conexões são realizadas com o nosso módulo principal escrito em Bluespec (`AvalonTop`) para permitir que este tenha acesso a todos os recursos necessários do barramento PCI Express:

- **AvalonSlave** é conectada ao **bar0** do PCI Express. Essa interface **recebe** requisições de memória, ou seja, comandos de leitura ou escrita de um valor localizado em um certo endereço. Isso ocorre pois dispositivos PCI Express são vistos pelo software como se fossem uma determinada região da memória do sistema. Toda comunicação com o hardware que for iniciada pelo software é realizada na forma de acessos a um ponteiro localizado nessa região. Em espaço de *kernel* no Linux, esse ponteiro pode ser obtido

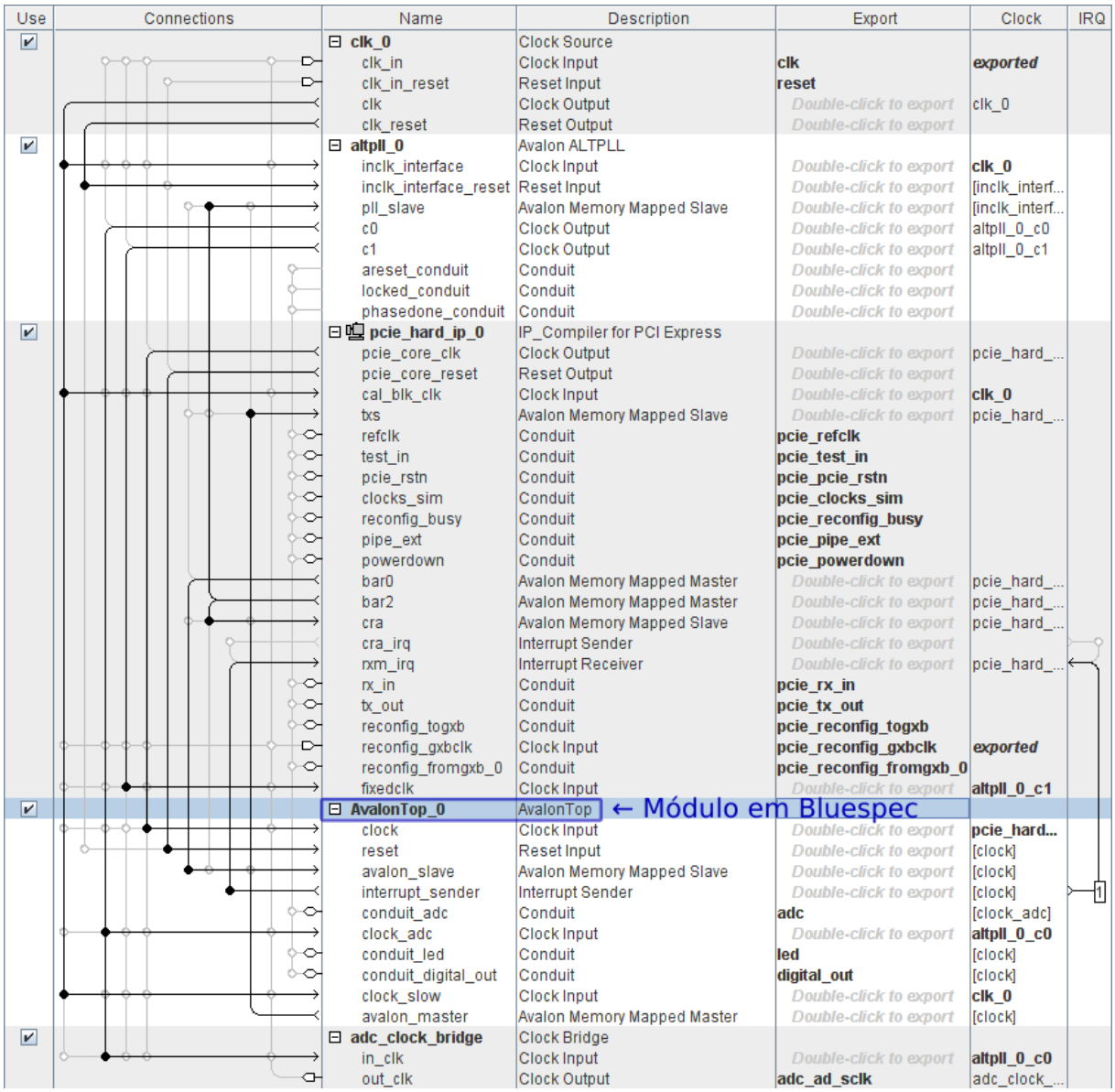

Figura 4.3- Conexões configuradas no Altera Qsys para permitir que o módulo de hardware principal escrito em Bluespec tenha acesso ao barramento PCI Express. A interface `AvalonSlave` do módulo, utilizada para receber comandos dos módulos de *kernel* e ferramentas de configuração, é conectada ao `bar0`. A interface `AvalonMaster`, por meio da qual o hardware pode escrever diretamente na memória RAM do computador (DMA), é conectada ao `txs`. A interface `InterruptSender`, utilizada pelo hardware para disparar interrupções a serem atendidas por uma rotina registrada por algum módulo de *kernel*, é ligada ao `rxm_irq`.

Fonte: Elaborada pelo autor.

com as rotinas `pci_resource_start` e `ioremap_nocache`. Em espaço de usuário, os dispositivos são mapeados em uma árvore de diretórios abaixo de `/sys/devices/pci*`, e um ponteiro pode ser obtido efetuando a chamada de sistema `mmap` sobre o arquivo `resource0` contido dentro do subdiretório correspondente ao dispositivo desejado.

- **AvalonMaster** é conectada ao **txs** (abreviatura de *transmission slave*) do PCI Express. Essa interface **transmite** requisições de memória ao barramento principal do PC. Ela é utilizada para obter acesso direto à memória (DMA, do inglês *direct memory access*), ou seja, realizar leituras ou escritas na memória RAM do computador sem intermediação do processador. Do ponto de vista do software, para empregar o recurso de DMA, é necessário chamar `pci_set_master` (em espaço de *kernel*) para dar permissão ao dispositivo de enviar requisições ao barramento. Além disso, uma região da memória RAM deve ser reservada (por exemplo, com a rotina `dma_alloc_coherent`) para que o dispositivo realize DMA sem correr o risco de sobrescrever endereços em uso por outros programas. O software deve informar o endereço físico dessa região ao hardware, escrevendo-o em algum endereço mapeado para o `AvalonSlave`.

- **InterruptSender** é conectada ao **rxm_irq** do PCI Express. Essa interface envia interrupções ao processador. Interrupções fazem com que um dos núcleos do processador pare a tarefa atualmente em execução e pule para uma rotina definida pelo sistema operacional. No RTAI, as interrupções são gerenciadas por um mecanismo denominado I-pipe, que as processa em domínio de tempo real, repassando-as somente depois ao Linux (caso necessário). Para registrar uma rotina de tempo real a ser executada quando ocorrer uma determinada interrupção, utiliza-se a função `rt_request_irq`.

Entretanto, como não temos acesso ao diagrama do circuito que realiza a conversão do barramento PCI Express para Avalon, efetuamos alguns testes a fim de garantir que esse processo não originaria comportamentos anormais de latência.

Escrevemos um módulo em Bluespec muito simples[65] que, quando recebia um comando do software por meio da interface `AvalonSlave`, imediatamente disparava uma interrupção pelo `InterruptSender`. Desenvolvemos também um módulo de *kernel* para o RTAI contendo uma

rotina periódica que, a cada 100 μs, lia o relógio do sistema com a função `rt_get_time_ns` e logo em seguida enviava um comando para o hardware disparar a interrupção. Na rotina do software que tratava a interrupção, líamos o relógio do sistema novamente e subtraíamos ambos os instantes para registrar a latência.

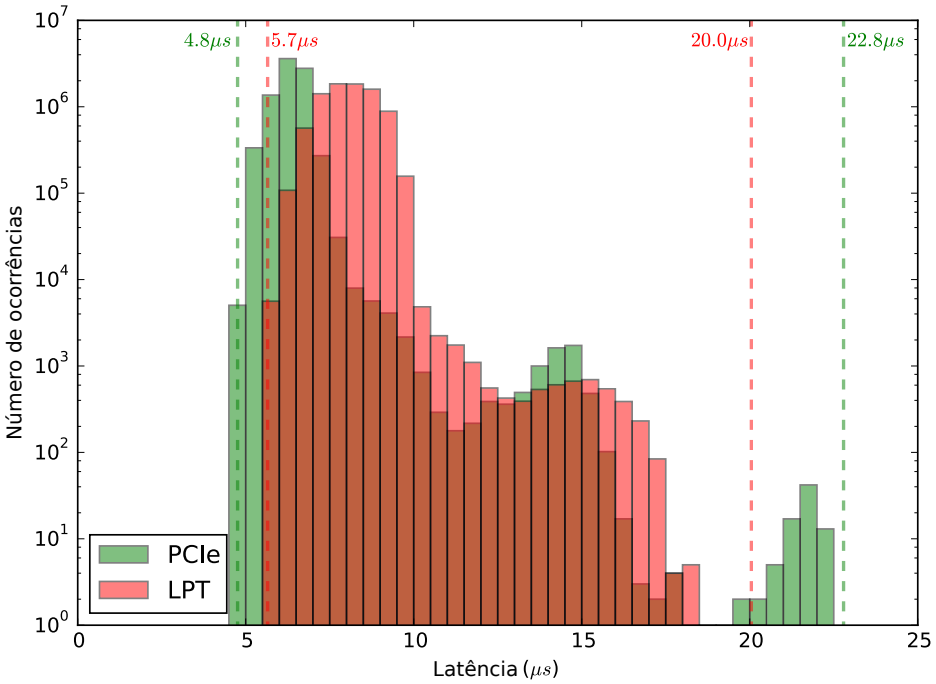

Figura 4.4- Histograma da latência de atendimento a interrupções do barramento PCI Express (em verde) e do porto paralelo (em vermelho) com RTAI, produzidas por um hardware externo como resposta a um comando enviado por uma tarefa periódica no domínio de tempo real. As medidas foram retiradas do mesmo computador utilizado para a confecção do discriminador.

Fonte: Elaborada pelo autor.

Como o computador utilizado neste trabalho possuía porto paralelo (LPT) nativo em sua placa-mãe, realizamos uma medida de latência da interrupção do LPT análoga à medida realizada para o PCI Express (PCIe). Inserimos no LPT um conector com o pino 2 (*data out* 0) em curto com o pino 10 (*acknowledge*). Para causar um disparo da interrupção da LPT, a rotina periódica do módulo de tempo real produzia uma borda de subida no nível lógico do primeiro bit de dados.

Forçamos carga de disco e de rede no computador e aferimos a latência das interrupções em ambos os barramentos. A Figura 4.4 apresenta um histograma dessas medidas. É importante notar que a escala vertical é logarítmica, e portanto barras que indiquem uma única ocorrência não aparecem no gráfico. Para suprir essa deficiência, plotamos linhas verticais demarcando a

latência mínima e a máxima de cada um dos barramentos. A latência mínima (4,8 μs) e a média (6,4 μs) do barramento PCIe foram inferiores às do LPT (5,7 μs e 8,1 μs, respectivamente), porém a latência máxima foi superior no PCIe (22,8 μs contra 20,0 μs). De qualquer forma, a variação de 2,8 μs é tolerável em nossa escala de tempo. Podemos considerar, assim, validado o comportamento da infraestrutura de comunicação via PCI Express utilizada no projeto.

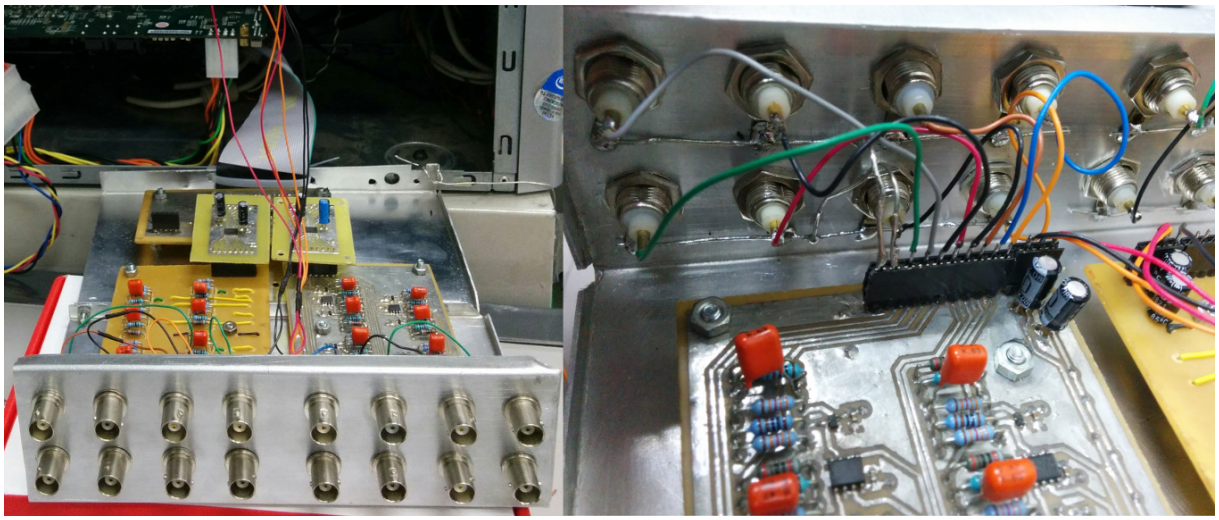

Figura 4.5- Painel do sistema de aquisição customizado construído como parte desta tese.
Fonte: Elaborada pelo autor.

4.3 Sistema de aquisição customizado

Algumas placas de aquisição disponíveis no mercado podem ser utilizadas em conjunto com o Analogy,[66] um pacote livre e de código aberto de *drivers* de tempo real. Entretanto, as placas-mães geralmente não suportam que um dispositivo PCI Express faça DMA nos endereços do barramento mapeados para outro dispositivo, ou seja, permitem DMA apenas na região da memória RAM. Devido a essa limitação, caso o sinal fosse adquirido por alguma dessas placas comerciais, ele teria obrigatoriamente que passar pelo software e ser reenviado para o FPGA antes de começar a ser processado pelo hardware dedicado. Nenhuma adaptação nos *drivers* seria capaz de mudar esse fato.

Optando-se por um sistema de aquisição customizado, conectado diretamente ao FPGA, remove-se completamente essa fonte adicional de latência: o hardware pode começar a processar o sinal em segundo plano, antes mesmo do software ter recebido uma interrupção

(sinalizando a disponibilidade de dados) e iniciado suas etapas de processamento.

A Figura 4.5 mostra o painel do sistema de aquisição construído como parte desta tese. Ele oferece 16 canais de entrada por meio de conectores BNC, mesmo padrão de conector adotado em sistemas comerciais. Desta forma, os cabos dos amplificadores conectados ao aquário podem ser facilmente alternados entre os sistemas de aquisição descritos na seção 2.1.

Nosso sistema de aquisição funciona, essencialmente, com base em dois conversores AD modelo MAX1280[43] operando em conjunto. A Figura 4.6 apresenta uma das placas de circuito impresso construídas para a montagem desses conversores. Apesar de imprimirmos trilhas com largura de somente 0,233 mm para possibilitar a solda do MAX1280, que possui encapsulamento TSSOP com espaçamento de 0,65 mm entre os terminais, foi possível fabricar as placas com o processo de transferência térmica de toner.

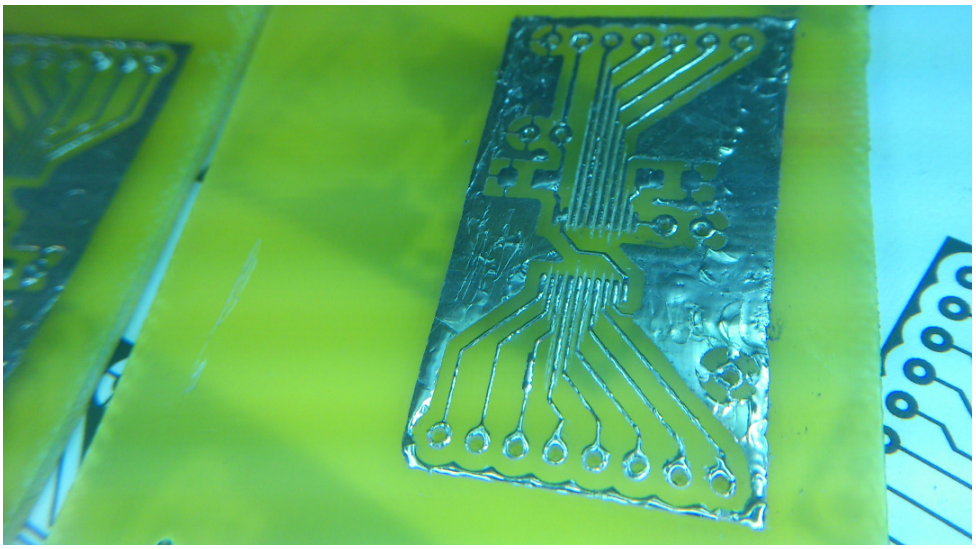

Figura 4.6- Fabricação da placa de circuito impresso para o conversor MAX1280. Foi possível utilizar o processo de transferência térmica de toner apesar do circuito integrado ter encapsulamento TSSOP com espaçamento de somente 0,65 mm entre os terminais.

Fonte: Elaborada pelo autor.

4.3.1 *Front-end* analógico

O MAX1280 é capaz de medir tensões de 0 a 2,5 V, faixa bastante comum entre os conversores unipolares atualmente disponíveis no mercado. A fim de proteger os conversores contra tensões fora da faixa tolerada (-0,3 a 2,8 V), e de permitir que nosso sistema seja eletricamente compatível com os sistemas de aquisição comerciais disponíveis no laboratório (seção 2.1), cuja

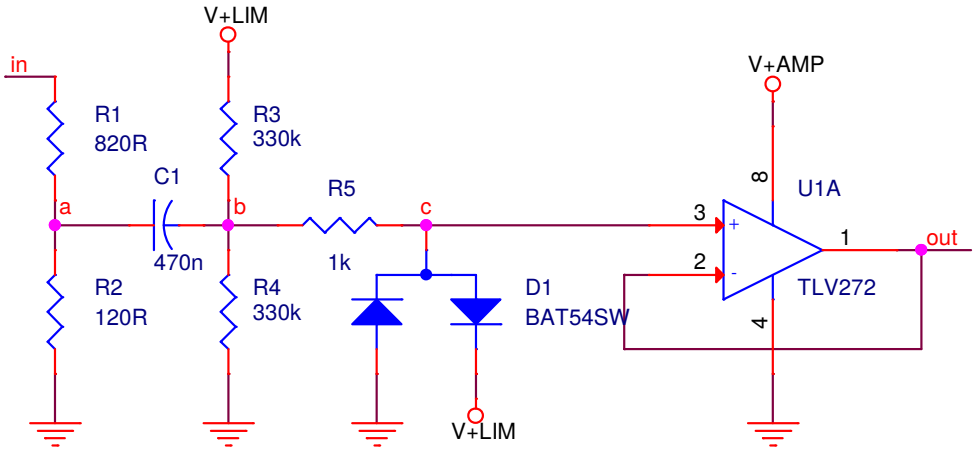

Figura 4.7- Diagrama do circuito de *front-end* analógico do sistema de aquisição customizado. O circuito divide a tensão de entrada por $^{940}/_{120} \approx 7{,}83$ (ponto a), acopla em corrente alternada com valor de repouso em $^{1}/_{2}V_{+\text{LIM}}$ (ponto b), limita a tensão resultante entre $-0{,}3$ V e $V_{+\text{LIM}} + 0{,}3$ V (ponto c) e, por fim, reforça o sinal com um seguidor.

Fonte: Elaborada pelo autor.

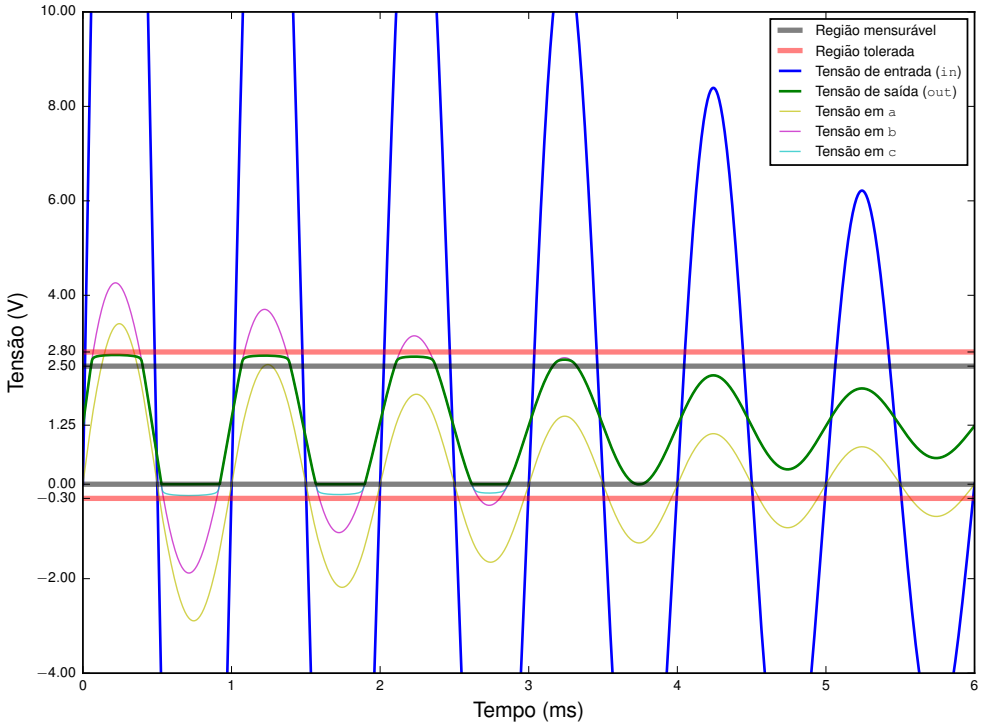

Figura 4.8- Simulação SPICE do *front-end* analógico com $V_{+\text{LIM}} = 2{,}5$ V. Mesmo quando a tensão de entrada (em azul) ultrapassa os 10 V previstos, o circuito é capaz de proteger o conversor AD, mantendo a tensão de saída (em verde) sempre dentro da região tolerada (em vermelho).

Fonte: Elaborada pelo autor.

faixa de operação é de -10 a 10 V, montamos um *front-end* analógico (diagrama esquemático na Figura 4.7), replicando-o para cada um dos 16 canais de entrada.

Após passar por um divisor de tensão e de ter sua componente DC deslocada para o centro da faixa de operação (1,25 V), o sinal é limitado à faixa tolerada pelo conversor por um diodo Schottky BAT54SW,[67] e por fim é reforçado por um seguidor construído com o amplificador operacional *rail-to-rail* TLV272.[68]

A Figura 4.8 mostra uma simulação SPICE[69] desse circuito, na qual fornecemos um sinal de entrada cuja velocidade de varrimento (*slew rate*) é da mesma ordem da observada durante os experimentos.

4.3.2 Módulo de controle dos conversores AD

O MAX1280 é comandado por um protocolo compatível com SPI (*Serial Peripheral Interface*), composto pelos sinais $\overline{\text{CS}}$ (*chip select*), SCLK (*serial clock*), DIN (*data input*) e DOUT (*data output*). Além desses pinos, que fazem parte da maioria dos componentes compatíveis com SPI, o MAX1280 conta também com uma saída SSTRB (*Serial-Strobe*), que é pulsada sempre um ciclo antes dos dados de conversão começarem a sair do pino DOUT. Mantemos o pino $\overline{\text{CS}}$ de ambos os conversores sempre em nível lógico baixo.

Nosso circuito de controle dos conversores (`DualAD`) está localizado em um domínio de *clock* de 6,4 MHz. Esse mesmo *clock*, que é gerado por uma das PLLs (*phase locked loops*) internas ao FPGA, comanda o sinal SCLK de ambos os MAX1280.

O sinal DIN de ambos os conversores também é comum, e é gerado por um circuito composto por um registrador de deslocamento de 8 bits (`ctrl`), que armazena o próximo byte de controle a ser enviado, um contador de 4 bits (`cnt`) que assegura o envio de um novo byte de controle a cada 16 ciclos, e um contador de 3 bits (`ch`), que armazena o canal atual. Sempre que `cnt` chega a seu valor máximo (15), um novo byte de controle gerado com base no canal `ch` é inserido em `ctrl`, e `ch` é incrementado. Cada bit de `ctrl` é enviado para DIN durante os ciclos em que $0 \leq \texttt{cnt} < 8$.

Como uma nova conversão é realizada a cada 16 ciclos (a 6,4 MHz), a taxa de amostragem total é de 400 kS/s, ou seja, 50 kS/s para cada um dos 8 canais de cada conversor.

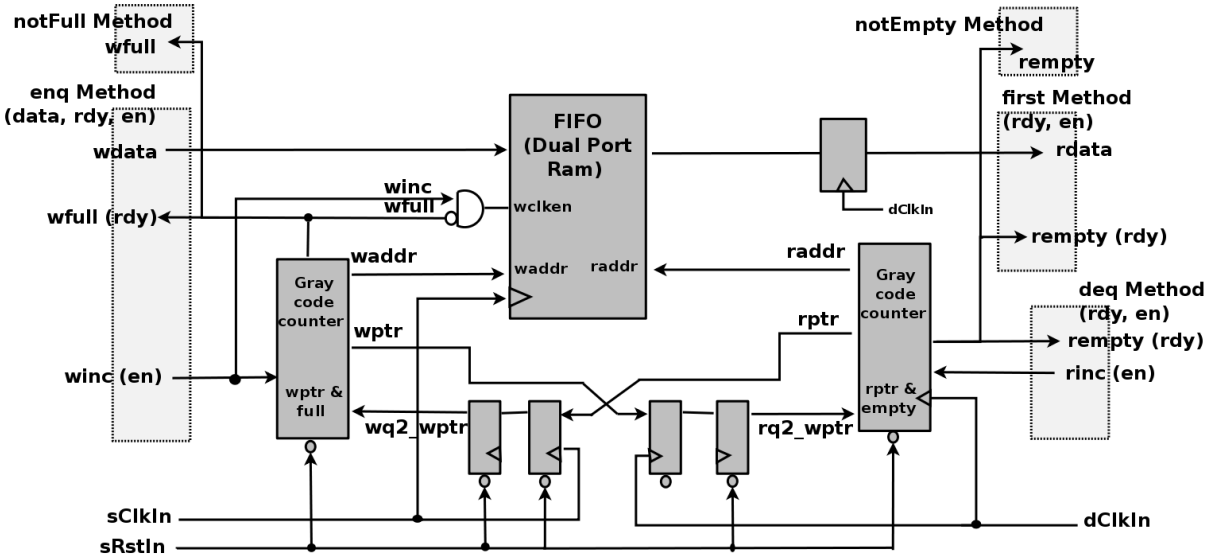

Figura 4.9- Circuito lógico de uma fila (FIFO – *first-in first-out*) sincronizadora da biblioteca padrão do Bluespec. Os dados são inseridos na fila em um domínio de *clock* e retirados em outro. O circuito implementa os cuidados necessários para com os ponteiros de cabeça e de cauda da fila, que são sincronizados por cadeias de 2 registradores seguidos (para evitar metaestabilidade) e utilizam um código Gray para impedir que atrasos diferentes na propagação de cada bit do contador corrompam seu valor ao cruzar os domínios de *clock*.

Fonte: Adaptado de BLUESPEC,[70] p. 389.

O sinal SSTRB é usado para garantir a sincronia do circuito de controle com ambos os conversores. Sempre que um dos conversores pulsa SSTRB, redefinimos o valor de `cnt` para 12, que equivale ao número de ciclos para que a conversão comece a ser lida de DOUT após o primeiro bit de comando ser enviado para os conversores.

Durante os ciclos em que $0 \leq \texttt{cnt} - 12 \pmod{16} < 12$, os sinais DOUT0 e DOUT1 (pinos DOUT de cada um dos conversores) são lidos e inseridos cada um em um registrador de deslocamento de 12 bits diferente (`shiftReg0` e `shiftReg1`).

Quando $\texttt{cnt} = 8 \equiv 24 \pmod{16}$, `shiftReg0` e `shiftReg1` contém as amostras provenientes de cada um dos conversores. Essas amostras são enviadas, juntamente com o número do canal, para uma FIFO de sincronização (Figura 4.9).

Já no domínio de *clock* de 125 MHz, a saída da FIFO de sincronização é passada por um serializador parcial da PAClib (`mkFunnel_Indexed`), que recebe como entrada ambas as amostras (lidas ao mesmo tempo dos dois conversores) e fornece como saída uma amostra por ciclo.

A saída final do módulo `DualAD` é um fluxo (`PipeOut`) de tuplas `ChSample`, contendo um número de canal de 4 bits e uma amostra de 12 bits. O bit mais significativo do canal indica

de qual dos dois conversores ele originou-se. Os outros 3 bits seguem a mesma convenção do conversor. A cada 2,5 μs (cerca de 312 ciclos a 125 MHz), duas dessas tuplas ficam disponíveis para serem retiradas da saída.

4.3.3 Infraestrutura de teste

A infraestrutura de teste relacionada ao sistema de aquisição é composta por dois módulos com funcionalidades aproximadamente inversas. Um deles (`ContinuousAcq`) permite adquirir dados diretamente dos conversores AD, pausando todo o *pipeline* de processamento de sinais do circuito. O outro (`MockAD`) interrompe o fluxo que parte dos conversores AD, substituindo por um fluxo lido da memória RAM do computador. Ambos serão discutidos a seguir.

Figura 4.10- Demonstração do modo de aquisição contínua de dados, que permite que o sistema seja utilizado como se fosse uma placa de aquisição comum, monitorando qualquer um dos 16 canais de entrada, mesmo que nem todos estejam habilitados para uso pelo discriminador.

Fonte: Elaborada pelo autor.

Módulo de aquisição contínua (ContinuousAcq)

Este módulo permite que o sistema seja utilizado como se fosse uma placa de aquisição simples. O software deve fornecer ao módulo o endereço físico de um *buffer* capaz de armazenar uma quantidade fixa (definida em tempo de compilação pela constante `ContinuousAcqBufSize`) de palavras de 64 bits. O módulo preenche com zeros à esquerda cada amostra proveniente do AD até completar 16 bits. Em seguida, as amostras são aglutinadas em conjuntos de 64-bits utilizando uma variação de um desserializador da PAClib (`mkClearableUnfunnel`) e escritas no *buffer* via DMA. Sempre que o *buffer* é preenchido até metade de seu nível total, uma interrupção avisa o *driver* para que este possa ler o conteúdo do *buffer* e repassá-lo ao aplicativo interessado.

A Figura 4.10 demonstra o funcionamente de um aplicativo de osciloscópio (`scope.py`) capaz de monitorar qualquer um dos 16 canais de entrada do sistema de aquisição com o auxílio deste módulo de hardware.

Módulo mimetizador de AD (MockAD)

Este módulo disponibiliza um comando que recebe o endereço físico de um *buffer* localizado na memória RAM. Toda vez que esse comando é chamado, uma quantidade fixa (definida em tempo de compilação pela constante `MockADBufSize`) de palavras de 64 bits é lida começando do endereço fornecido. Essas palavras são quebradas em pedaços de 16 bits por um serializador parcial da PAClib (`mkFunnel`). Por fim, os pedaços são individualmente truncados para a resolução utilizada pelo AD (no projeto atual, 12 bits) e destinados ao mesmo fluxo do *pipeline* que seria seguido se esses dados estivessem saindo do AD.

Antes de utilizar o módulo, o multiplexador localizado na saída do AD deve ser alternado de posição, modificando via software o valor do registrador `adcMocked` (vide Figura 4.2). É importante notar que este é o único módulo do sistema que realiza leituras via DMA. Todos os outros módulos que utilizam DMA fazem-no para operações de escrita, para as quais o barramento não produz nenhuma resposta. Portanto, este módulo é conectado diretamente à linha de respostas do `AvalonMaster` (como pode ser observado na Figura 4.2).

Como a lógica do *pipeline* de processamento de sinais após a saída do AD é insensível à

latência, o `MockAD` não fica restrito à mesma taxa de saída de dados que o `DualAD` (2 tuplas `ChSample` a cada 2,5 μs). Teoricamente, poderíamos atingir taxas até 16$\times$ superiores a essa, considerando que o processamento de uma janela ocorra em média em 500 μs e que cada peixe dispare DOEs a 60 Hz.

Incluímos em nosso pacote de software de tempo real um módulo de *kernel* simples (denominado `gymnort_mockad`) para fornecer ao `MockAD` um arquivo gravado de um experimento. Apesar de atualmente verificarmos se o *buffer* terminou de ser processado por meio de *polling*, ou seja, utilizando um laço que inspeciona continuamente se o módulo `MockAD` está ocupado, foi possível processar séries de dados previamente gravadas em um tempo $(10{,}3 \pm 0{,}8)$ vezes inferior à duração da gravação.

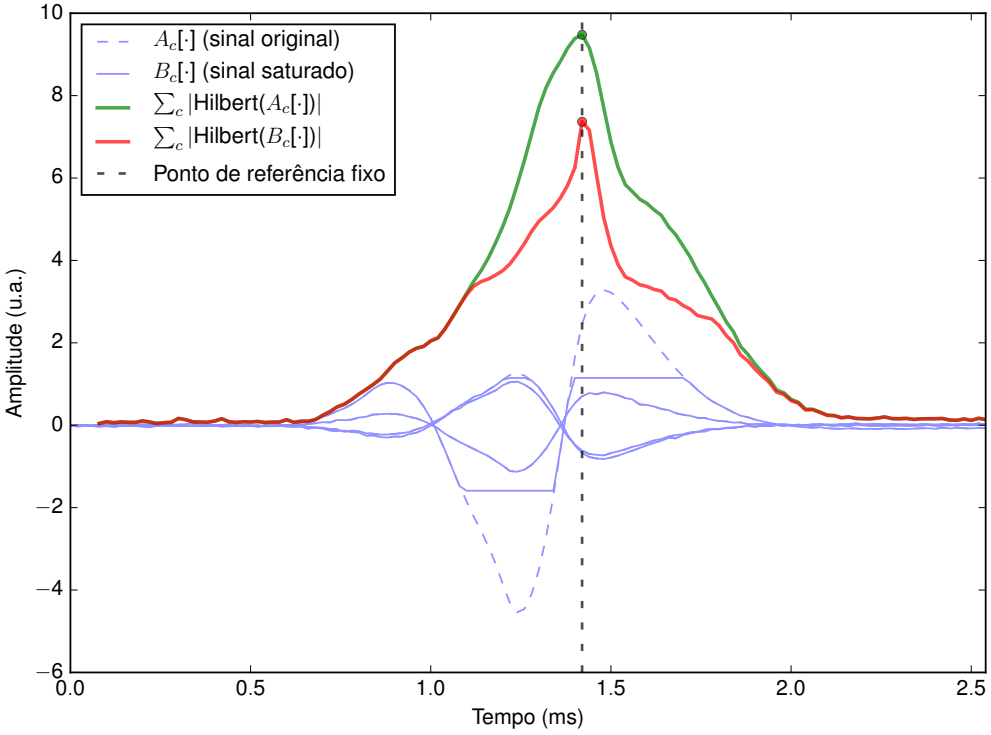

Figura 4.11- Soma dos módulos da transformada de Hilbert (aproximada com filtro IIR) de uma DOE em 4 canais, para um sinal bem-formado A_c e para uma versão saturada B_c da mesma DOE. Sabe-se[8] que o pico da soma de transformadas de Hilbert cai em um instante de referência aproximadamente fixo dentro das DOEs de um peixe. Esse comportamento do pico mantém-se resiliente à saturação de um dos canais.

Fonte: Elaborada pelo autor.

4.4 Módulo delimitador de janelas

O algoritmo de segmentação utilizado na implementação *off-line* (seção 3.2.1) é inadequado para uma implementação de tempo real, por depender da resposta de um filtro FIR com latência de 5 amostras, além de esperar um período de histerese de $N_b = 16$ amostras para chegar à conclusão que uma janela chegou ao fim. A latência introduzida pela necessidade de aguardar a conversão dessas 21 amostras (a 50 kS/s) é de 420 μs, que não pode ser tolerada, pois representa quase metade do requisito de tempo de 1 ms.

Para solucionar esse problema, utilizamos um resultado de Jun, Longtin e Maler[8] que indica que o pico da somatória do módulo das transformadas de Hilbert em um aquário multieletrodos é capaz de apontar para um ponto de referência fixo dentro de uma DOE (segundo os autores, com erro da ordem de apenas 20 μs), independente da posição do peixe dentro do aquário. A Figura 4.11 ilustra o resultado desse cálculo para um conjunto de 4 canais com sinal íntegro e para o mesmo conjunto com sinal saturado, demonstrando que esse método é resiliente, também, à saturação de um dos canais do aquário.

Aproximamos o valor da componente imaginária da transformada de Hilbert com um filtro de resposta ao impulso infinita (IIR, do inglês *infinite impulse response*) de ordem 4. Apesar desse filtro apresentar uma latência de 4 amostras, esta não influencia na latência final para detecção do término da DOE, pois o pico da somatória está geralmente localizado cerca de 700 μs (35 amostras a 50 ks/S) antes desse término. Sendo assim, o pico pode ser detectado muito antes do término da DOE, permitindo que o instante de ocorrência do fim da janela seja previsto com antecedência.

Como o conversor AD introduz no *pipeline* um máximo de 2 amostras a cada 2,5 μs, podem ser gastos em média 156 ciclos (a 125 MHz) processando cada amostra. Realizando uma única multiplicação por ciclo, um filtro IIR de ordem 4 pode ser implementado em cerca de 10 ciclos. Como esse valor é muito inferior aos 156 ciclos disponíveis, resolvemos adotar a estrutura de um único multiplicador, causando uma pausa (*stall*) no *pipeline* durante o tempo em que uma amostra está sendo processada pelo filtro. Essa estrutura é denominada *folded pipeline* (ou seja, *pipeline* dobrado) na terminologia da PAClib. O multiplicador utilizado foi parametrizado para processar dois valores de ponto fixo, cada um com 16 bits de parte inteira

e 16 bits de parte fracionária.

Uma vez calculada a parte imaginária da transformada por meio do filtro IIR, esta é arredondada para um inteiro de 16 bits. Em seguida, esse valor é elevado ao quadrado, e somado com o quadrado do valor do sinal atrasado de 4 amostras (parte real da transformada). Por fim, calculamos a raíz quadrada do valor de 32 bits resultante com o auxílio do módulo `mkSquareRooter` da biblioteca padrão do Bluespec, operação que leva 18 ciclos (também muito abaixo dos 156 ciclos disponíveis).

O módulo da transformada de Hilbert é acumulado conforme os valores de cada canal vão tornando-se disponíveis. Quando a soma de toda uma varredura de canais é concluída, o módulo `WindowMaker` incrementa o contador de *timestamp* (`curTs`) e atualiza uma máquina de estados destinada a determinar o início e o fim das janelas.

A detecção de início de janelas ocorre de forma similar à apresentada na seção 3.2.1 (limiar com uma histerese $N_b = 4$). Entretanto, em lugar do sinal $I[\cdot]$ representar a soma de quadrados dos canais filtrados, ele representa agora a soma dos módulos das transformadas de Hilbert. A detecção de fim é limitada pela distância ao pico de $I[\cdot]$, ou seja, caso uma tendência de novo pico (valor da derivada de $I[\cdot]$ positivo e acima de um limiar) não seja observada durante um intervalo de 35 amostras, a janela é finalizada.

Sempre que o fim de uma janela é detectado, o módulo `WindowMaker` produz como saída um marcador de fim (`EndMarker`) acompanhado de uma estrutura `WindowInfo` que contém as seguintes informações:

- `timestamp`: instante de ocorrência do final da janela, em número de amostras;
- `size`: comprimento da janela, ou seja, número de amostras que se passaram desde o início da janela até então;
- `reference`: número de amostras que se passaram desde o instante em que o pico da soma dos módulos das transformadas de Hilbert foi detectado.

4.5 Módulo de transferência de janelas

Este módulo (denominado `WindowDMA`) é responsável por transferir em segundo plano, via DMA, todas as amostras pertencentes a janelas que forem detectadas pelo módulo delimitador (`WindowMaker`), e em seguida sinalizar ao software o término dessa transferência por meio de uma interrupção.

Com a finalidade de evitar que o barramento principal do computador seja ocupado por requisições DMA desnecessárias, este módulo possui um *buffer* circular interno, composto por blocos de memória do FPGA, que armazena as amostras de sinal até o momento em que um marcador de fim de janela (`EndMarker`) é recebido.

Uma vez recebido o marcador de fim, o número de amostras adequado (de acordo com o campo `size` da estrutura `WindowInfo`) é retirado do *buffer* interno e enviado em uma única rajada de DMA para a memória RAM do computador. Essa rajada demora no máximo 10 μs, tempo inferior a uma varredura completa de canais (20 μs). Concluída a transferência, o módulo dispara uma interrupção.

A fim de garantir que não sejam perdidos dados caso uma janela seja detectada enquanto a anterior ainda estiver sendo processada pelo software, é utilizado um *buffer* duplo. Ou seja, este módulo dispara a rajada iniciando ora no começo do *buffer*, ora no meio do *buffer*, de forma a nunca sobrescrever a janela imediatamente anterior à atual.

4.6 Módulo de sobreposição de disparos

O módulo de sobreposição de disparos, também chamado de módulo de minimização de distâncias (`DistMinimizer`), é destinado a resolver a equação 3.2.6, encontrando as posições iniciais s_1 e s_2 dos disparos de cada peixe dentro de um segmento de sinal.

Este é o módulo ao qual foi dada a maior prioridade de implementação. Logo no início do desenvolvimento de nosso discriminador em tempo real, construímos um protótipo deste módulo, a fim de aferir a viabilidade do projeto. Essa prioridade deve-se ao fato de que, como mostramos na Figura 3.5, esta é a etapa do algoritmo de discriminação que consome o maior tempo de execução. Entretanto, trata-se de uma tarefa que possui um alto grau de paralelismo

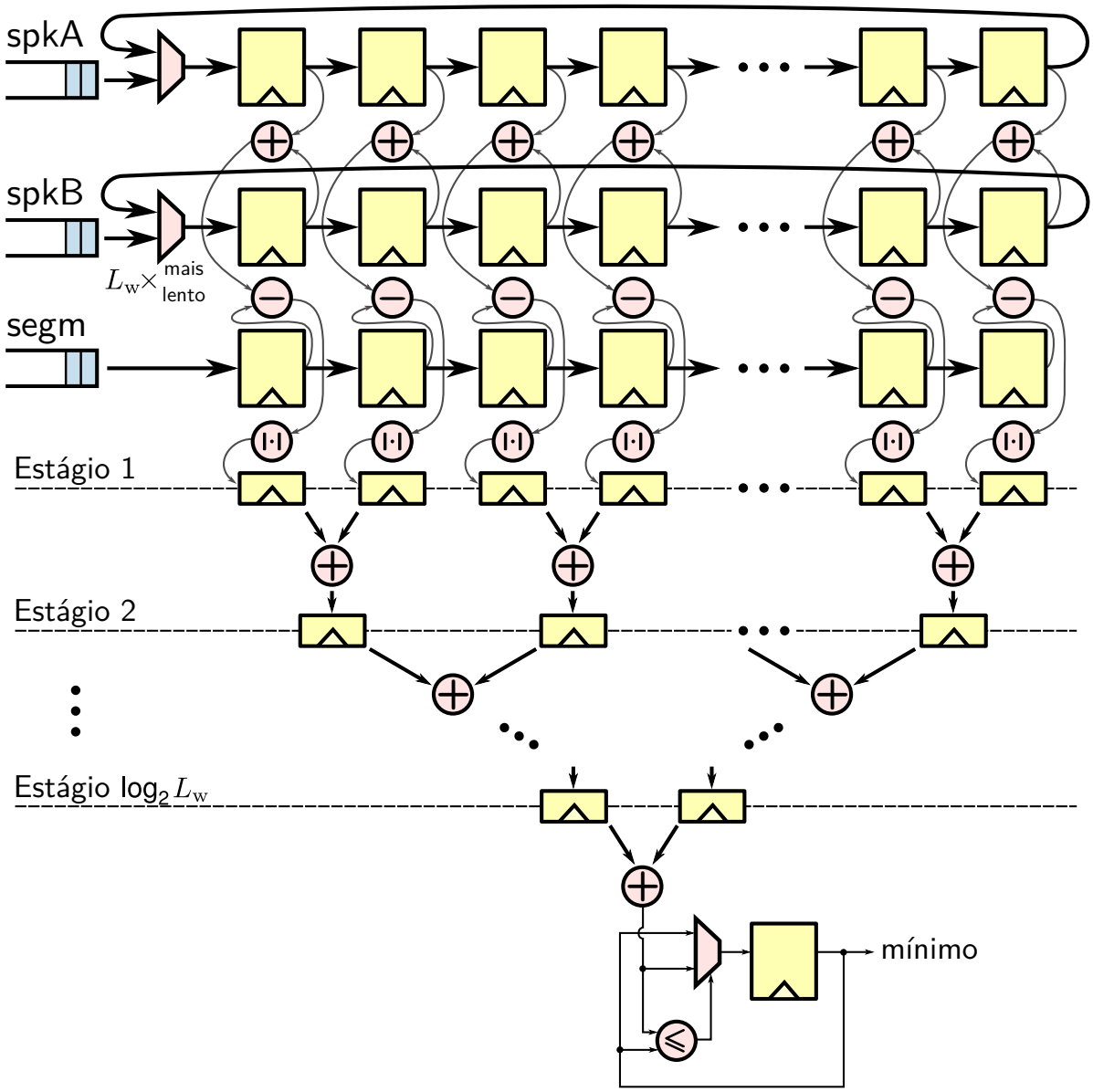

Figura 4.12- Diagrama simplificado (*data-path*) da arquitetura de hardware do módulo de sobreposição de disparos (`DistMinimizer`). As DOEs previamente emitidas pelos peixes A e B são fornecidas ao circuito pelas filas `spkA` e `spkB`. O segmento de sinal atualmente em análise é fornecido pela fila `segm`. Uma vez alimentadas as cadeias de L_w registradores, os multiplexadores conectados às filas alternam para a entrada superior, e o circuito funciona durante $L_w^2 + \log_2 L_w$ ciclos. Ao longo desse tempo, a cadeia de registradores `spkB` rotaciona as amostras a uma velocidade L_w vezes menor que a cadeia `spkA`. Somamos os valores contidos na cadeia `spkA` com os da `spkB`, subtraímos dos valores da cadeia `segm`, e por fim calculamos o módulo. Os valores de módulo passam por uma árvore de somadores organizados em um *pipeline* de $\log_2 L_w$ estágios. O valor mínimo encontrado ao longo de todo esse processo é armazenado em um registrador, juntamente com o número do ciclo no qual este foi obtido, o que permite recuperar as posições para as quais os sinais haviam sido deslocados naquele instante.

Fonte: Elaborada pelo autor.

com granularidade fina, portanto bastante adequada à implementação em FPGAs.

A Figura 4.12 apresenta um diagrama simplificado da arquitetura de hardware para um único canal, com janela de comprimento L_{w}. O funcionamento básico da arquitetura é dado por três cadeias de L_{w} registradores que operam de modo similar a um registrador de deslocamento (*shift-register*) — a cada ciclo, a amostra contida em um registrador pode ser movida para o registrador à direita. Duas dessas cadeias são circulares, ou seja, o último registrador pode realimentar o primeiro registrador da cadeia.

Inicialmente, essas cadeias de registradores são alimentadas por meio de filas do tipo *first-in first-out* (FIFOs), representadas em azul na figura:

- As FIFOs `spkA` e `spkB` recebem o sinal da última DOE previamente reconhecida como sendo emitida pelo peixe A e pelo peixe B, respectivamente. Esse processo é disparado pelo software de tempo real por meio de um canal de realimentação (*feedback*) após consolidada a classificação de cada janela processada.

- A FIFO identificada na figura pelo nome `segm` recebe, do módulo delimitador de janelas, o segmento de sinal a ser analisado e reconhecido.

Sempre que alguma cadeia está sendo alimentada, os multiplexadores posicionados na saída das FIFOs correspondentes permanecem com sua entrada inferior ativada.

Uma vez alimentadas as cadeias, ambos os multiplexadores conectados a elas são alternados para a entrada superior. A cada ciclo de relógio, a cadeia de registradores `spkA` é rotacionada de uma amostra. Já a cadeia `spkB` rotaciona a uma velocidade mais lenta, sendo movimentada somente a cada L_{w} ciclos. A cadeia `segm` permanece estática nesta fase de funcionamento do circuito.

A todo momento, uma lógica combinacional soma o valor contido em cada registrador da cadeia `spkA` com aquele armazenado no registrador logo abaixo, pertencente à cadeia `spkB`. O resultado da soma é subtraído do valor contido no registrador mais abaixo, membro da cadeia `segm`. Calculamos, então, o módulo do resultado de cada subtração, alimentando um conjunto de registradores que formam o primeiro estágio de um *pipeline* de $\log_2 L_{\mathsf{w}}$ estágios. Cada estágio do *pipeline* forma um nível de uma árvore de somadores, até que seja obtido o

valor da soma de todos os módulos. Toda a aritmética envolvida nesse processo é saturante, ou seja, caso o valor resultante de alguma operação supere o máximo representável com o número de bits do barramento em questão, o valor de saída é truncado em vez de sofrer *wrap-around* (voltar a ser contado a partir do outro extremo da faixa numérica aceitável). Desta forma, o uso do circuito é seguro mesmo quando os sinais de entrada apresentam saturação.

O circuito realiza esse processo durante, ao todo, $L_{\mathsf{w}}^2 + \log_2 L_{\mathsf{w}}$ ciclos. Esse tempo é suficiente para que todas as posições iniciais possíveis de `spkA` e `spkB` sejam avaliadas, e para que as últimas somas propaguem-se até o final do *pipeline*. Um registrador armazena qual foi a menor soma total obtida ao longo de todo o processo, além do número do ciclo no qual essa soma foi obtida. O número do ciclo permite recuperar a informação de quais posições de `spkA` e `spkB` estavam sendo analisadas no momento em que o valor mínimo foi encontrado.

O mesmo circuito é generalizado para vários canais da seguinte forma: as cadeias de registradores e a árvore de somadores são replicadas uma vez para cada canal (N_{ch} vezes). A saída de cada árvore de somadores alimenta uma nova árvore, cuja entrada recebe $\lceil \log_2 N_{\mathsf{ch}} \rceil$ valores. O circuito que registra o mínimo é movido para a saída dessa nova árvore. Como o número de estágios total do *pipeline* composto por ambas as árvores de somadores é de $\log_2 L_{\mathsf{w}} + \lceil \log_2 N_{\mathsf{ch}} \rceil$, o circuito deve ser monitorado por $L_{\mathsf{w}}^2 + \log_2 L_{\mathsf{w}} + \lceil \log_2 N_{\mathsf{ch}} \rceil$ ciclos para determinar o mínimo global da distância.

É importante notar que nosso circuito calcula o módulo das diferenças em vez de calcular o quadrado das mesmas, como faria nossa implementação em software. Assim, podemos dizer que se trata de um circuito de minimização da distância Manhattan, e não da distância euclidiana. Contudo, nossos ensaios demonstraram que a distância Manhattan é uma métrica tão boa para a tarefa em questão quanto a distância euclidiana. Assim, este módulo pode ser inteiramente implementado com elementos lógicos comuns de um FPGA, permitindo que os elementos multiplicadores (que seriam utilizados para calcular o quadrado) sejam economizados, o que pode facilitar a realização de trabalhos futuros interessantes (seção 6.2).

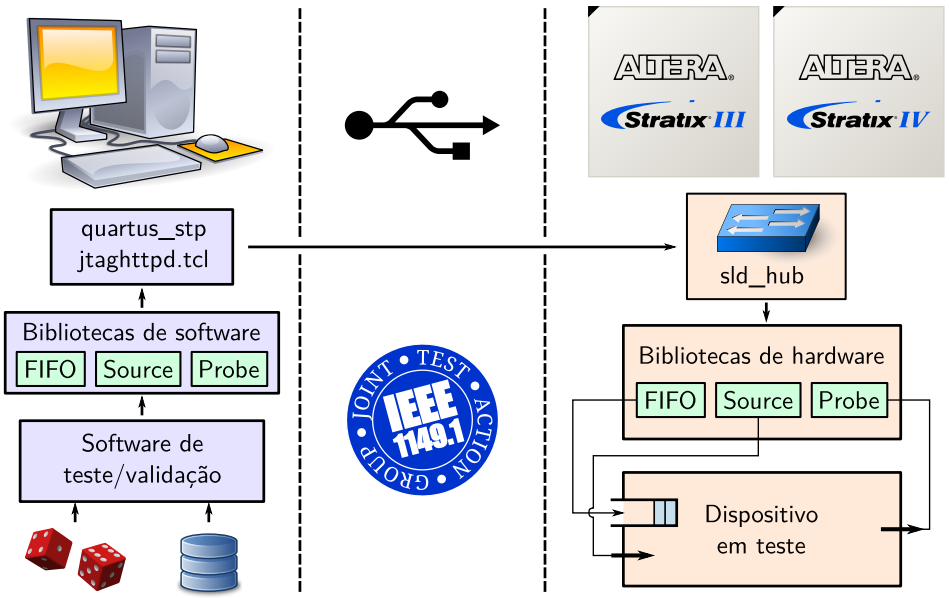

Figura 4.13- O software de teste e validação tem acesso ao hardware por meio de bibliotecas padronizadas, escritas em linguagem Python, contendo rotinas que mapeiam uma-a-uma para as operações correspondentes nos blocos de hardware, descritos em Bluespec. A comunicação entre o computador e o FPGA é realizada por meio do barramento JTAG, acessível na maioria dos kits de desenvolvimento através de uma interface USB. A ferramenta `quartus_stp` faz a interface entre o software em Python e o barramento JTAG. No lado do hardware, o bloco `sld_hub` distribui a comunicação JTAG para os diversos circuitos que compõe as bibliotecas de hardware.

Fonte: Elaborada pelo autor.

4.7 Ferramentas para teste e depuração de hardware

É bastante comum que projetos de hardware sejam testados por meio de simuladores. Essa abordagem é muito flexível, uma vez que a maioria das linguagens de descrição de hardware fornece algum mecanismo para chamar rotinas de software de dentro do circuito a ser simulado. Essas rotinas podem ser implementadas, por exemplo, por bibliotecas em linguagem C, dando liberdade ao projetista para gerar casos de teste bem elaborados.

Entretanto, por mais que as simulações de circuitos digitais por meio de ferramentas de software estejam cada vez mais confiáveis, estas nem sempre são suficientes para testar e validar um projeto. Não seria possível testar e depurar o sistema descrito nesta tese como um todo em um simulador, pois um modelo dos ADs não estaria disponível nas ferramentas de simulação. Em outros projetos, podem existir ainda outras barreiras para o uso de simuladores. Por exemplo, o tempo de simulação em um computador pode ser proibitivo devido à complexidade do circuito, exigindo que os testes sejam executados diretamente nos FPGAs.

Portanto, a comunicação entre computadores e FPGAs é uma ferramenta de grande valor

para o teste de projetos de hardware,[71,72] uma vez que preserva a flexibilidade na confecção de casos de teste, e também para a depuração, já que reduz a necessidade de utilizar equipamentos específicos, tais como analisadores lógicos. Padrões como o SCE-MI[73] foram desenvolvidos pela indústria com a finalidade de atender a essa demanda. Infelizmente, porém, a maioria dos kits de desenvolvimento da Terasic Inc. (com FPGAs da Altera), os de mais fácil obtenção no Brasil, ainda não possuem um bom suporte em suas ferramentas a esse padrão.

Como a metodologia de teste não era o foco central de nosso projeto, e implementar suporte a SCE-MI exigiria desenvolver e adaptar um hardware de comunicação distinto para cada kit de desenvolvimento com o qual resolvêssemos experimentar, optamos por desenvolver uma ferramenta simples que fizesse uso do barramento JTAG,[61] presente em todos os FPGAs da Altera.[74] Apesar de mais lenta que o SCE-MI, que geralmente utiliza dutos rápidos de comunicação (como o PCI Express), essa solução é portável e pode ser imediatamente utilizada em todos os kits de desenvolvimento que contenham FPGAs da Altera. Além disso, em kits que possuam PCI Express, como o adotado no projeto abordado por esta tese, nossa solução baseada em JTAG evita que testes pontuais interfiram ou conflitem com a utilização do PCI Express pelos demais módulos do projeto.

O funcionamento de nossa solução é ilustrado na Figura 4.13. Um software de teste e validação acessa o hardware por meio de bibliotecas de software padronizadas e orientadas a objetos. As bibliotecas foram implementadas em linguagem Python, e são bastante convenientes inclusive para a depuração do hardware em um terminal de comandos interativo. Cada chamada a um método pertencente a um objeto da biblioteca de software possui uma correspondência direta com alguma operação do bloco de hardware, implementado em linguagem Bluespec, que esse objeto representa. Publicamos a solução completa como software livre,[60] visando permitir a utilização desta por outros projetos.

4.7.1 Interceptação de dados trocados entre módulos do *pipeline*

Um recurso de nossa solução que facilita muito o seu uso para a depuração de circuitos é o fato de podermos instanciar um módulo que se comunique via JTAG em qualquer ponto do projeto, sem necessitar roteá-lo manualmente até o módulo principal. Esse recurso é

herdado da megafunção `altsource_probe` (da própria Altera), que usamos como base para a implementação de nossas bibliotecas de JTAG em Bluespec.

```
module [Module] mkWindowDMABuffer#(PipeOut#(OutItem) pipeIn)
                                   (PipeOut#(DMABufItem));
    /* ... */
    FIFOF#(DMABufItem) fifoOut <- mkFIFOF;

    Put#(DMABufItem) jtagTap <- mkJtagPut("TAP0",
                                          mkSizedFIFOF(4096));
    /* ... */

    (*hide*) let m <- mkTap(jtagTap.put,
                            f_FIFOF_to_PipeOut(fifoOut));
    return m;
    //return f_FIFOF_to_PipeOut(fifoOut);
endmodule
```

```
import jtaghttpclient, jtagfifo, pickle
jtag = jtaghttpclient.JTAGServer()

tap0 = jtagfifo.JTAGOutputFIFO(jtag[1][1]['TAP0'])

def savefifo(fifo, filename):
    a = []
    try:
        while True:
            a.append(fifo.deq())
    except: pass

    with open(filename, 'w') as f:
        pickle.dump(a, f)

savefifo(tap0, 'tap0.pickle')
```

Figura 4.14- Exemplo de uso das bibliotecas JTAG para depuração do hardware desenvolvido nesta tese. Á esquerda, apresentamos o código do hardware (em Bluespec) e, à direita, o código do software (em Python). Para interceptar os resultados do módulo `mkWindowDMABuffer`, inserimos um módulo `mkTap` em sua saída, que insere todo dado trafegado por esse caminho na fila de 4096 elementos associada ao ponto de medida "`TAP0`". O software instancia uma conexão com essa fila de saída e lê todo o seu conteúdo em um vetor, salvando os dados coletados em um arquivo "`tap0.pickle`".

Fonte: Elaborada pelo autor.

A Figura 4.14 mostra, ao lado esquerdo, um código em Bluespec no qual uma FIFO de 4096 elementos é instanciada e conectada a uma terminação JTAG denominada "`TAP0`". O hardware pode inserir elementos nessa FIFO por meio da interface `jtagTap`. O método `put` dessa interface é passado ao módulo `mkTap` da PAClib, que intercepta todos os dados que passarem por esse ponto do *pipeline*, inserindo-os na FIFO. Ao lado esquerdo, um código em Python chama repetidamente o método `fifo.deq()`, retirando todos os elementos da FIFO, e salvando-os, em seguida, no arquivo "`tap0.pickle`".

Essa técnica de interceptação de dados trocados entre módulos do *pipeline* foi muito importante ao longo do projeto, permitindo isolar quais módulos eram os responsáveis por causar determinados erros ou condições inesperadas. Esse processo é análogo a utilizar um osciloscópio ou analisador lógico para observar o comportamento de pontos intermediários de um circuito macroscópico.

4.7.2 Teste do módulo de sobreposição de disparos

Logo no início do projeto, construímos uma implementação preliminar (para um único canal) do módulo de sobreposição de disparos (seção 4.6), a fim de demonstrar a viabilidade de nossa proposta. Como nenhum outro módulo havia sido implementado naquele estágio do projeto, a solução de teste de hardware por meio do barramento JTAG foi essencial para permitir que uma prova de conceito fosse rapidamente realizada. Efetuamos uma bateria de testes nessa implementação para verificar que esta comportava-se como especificado.

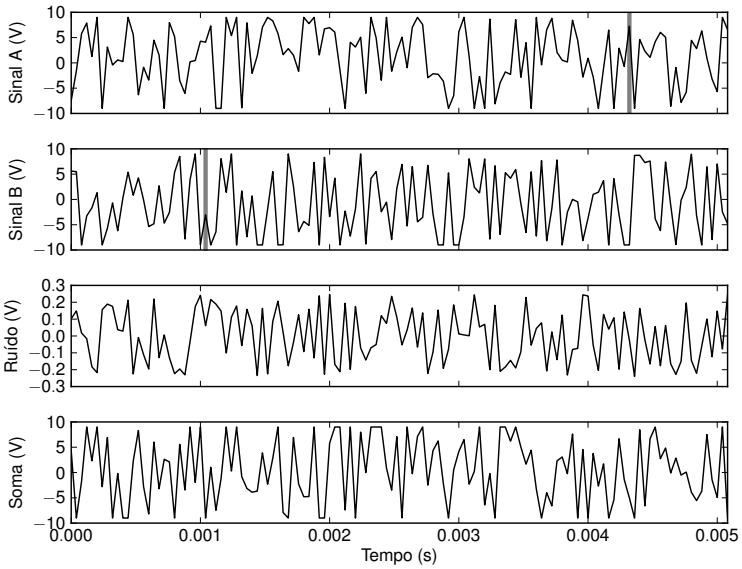

Figura 4.15- Geramos aleatoriamente dois sinais (A e B), que permitimos que saturem em alguns pontos, além de um ruído de amplitude 2 a 4× superior àquela observada nos experimentos. O software de testes escolhe duas posições aleatórias, mostradas no gráfico como barras verticais cinzas, que passam a ser o novo início dos sinais A e B quando estes são deslocados. A soma dos sinais deslocados e do ruído produz o sinal mostrado no gráfico mais abaixo. Enviamos ao FPGA somente os sinais A e B originais (antes de serem deslocados) e o sinal de soma, e verificamos que o circuito é capaz de recuperar as posições de deslocamento.

Fonte: Elaborada pelo autor.

Um contador foi acrescentado ao circuito para travar a movimentação das cadeias de registradores depois de passados $L_{\mathsf{w}}^2 + \log_2 L_{\mathsf{w}}$ ciclos de relógio, até que um comando externo fosse recebido pela interface JTAG indicando o início do próximo caso de teste.

Primeiro, montamos manualmente alguns casos de teste refletindo situações extremas que o circuito poderia enfrentar. Por exemplo, um segmento de sinal contendo ambos os pulsos (de A e de B) iniciando logo na primeira amostra; outro, com ambos os pulsos posicionados na última amostra do segmento. Verificamos que as posições dos pulsos eram corretamente

detectadas pelo circuito. Construímos também alguns casos de teste com base em dados reais, exatamente da forma como foram coletados nos experimentos, e constatamos que o circuito era capaz de discriminá-los tão bem quanto a nossa implementação em software.

Por último, passamos à execução intensiva de testes produzidos automaticamente, de forma aleatória. O gerador de números pseudoaleatórios adotado foi o Mersenne Twister MT19937.[46] Como ilustrado na Figura 4.15, geramos dois sinais (A e B) com uma distribuição uniforme no intervalo $[-12, 12]$, e em seguida truncamos as amostras para o intervalo $[-10, 10]$, mimetizando um comportamento de saturação. Desta forma, geramos sinais nos quais é comum observar um certo número de amostras saturadas. É importante lembrar que o circuito trabalha com números inteiros, e portanto, antes de serem enviados ao FPGA, esses valores de tensão são sempre convertidos para o intervalo $[-2^{11}, 2^{11} - 1]$, correspondente à mesma resolução de nosso sistema de aquisição (12 bits).

Deslocamos os sinais A e B para duas posições escolhidas ao acaso e, em seguida, somamos os sinais entre si e também a um ruído branco. O sinal resultante é mostrado no último gráfico da figura. As barras cinzas verticais dos dois primeiros gráficos indicam o novo início dos sinais A e B após estes sofrerem o deslocamento. Conhecendo somente os sinais originais A e B, antes de serem deslocados, é difícil identificar a olho nu o instante de tempo onde os mesmos começam dentro do sinal de soma. No entanto, quando os dados são fornecidos ao circuito, este é capaz de localizar corretamente as posições de início.

Geramos, ao todo, 30 mil casos de teste aleatórios. Cada conjunto de 10 mil foi transmitido para um kit de desenvolvimento FPGA distinto (dentre DE2, DE3 e DE4). Todos os sinais foram reconhecidos com sucesso. Depois de cada teste, verificamos não somente que as posições iniciais de A e B eram as mesmas que aquelas que haviam sido sorteadas pelo software, mas também que o valor mínimo da distância Manhattan encontrado pelo hardware era exatamente igual à soma dos módulos do sinal de ruído.

4.8 Otimização dos módulos em software

Esta seção descreve como as etapas de maior custo computacional do algoritmo de discriminação, exceto pela etapa de sobreposição de disparos (que foi implementada em hardware), foram

otimizadas para melhorar seu desempenho com relação à implementação *off-line* original.

Focamos nosso esforço de otimização em permitir que as rotinas em software pudessem fazer uso do conjunto de instruções AVX,[33] que está disponível nos processadores Intel desde a arquitetura Sandy Bridge, popularmente conhecida como "Intel Core de segunda geração". Essas instruções permitem realizar uma mesma operação de ponto flutuante em paralelo sobre vetores com 4 elementos de precisão dupla, ou com 8 elementos de precisão simples.

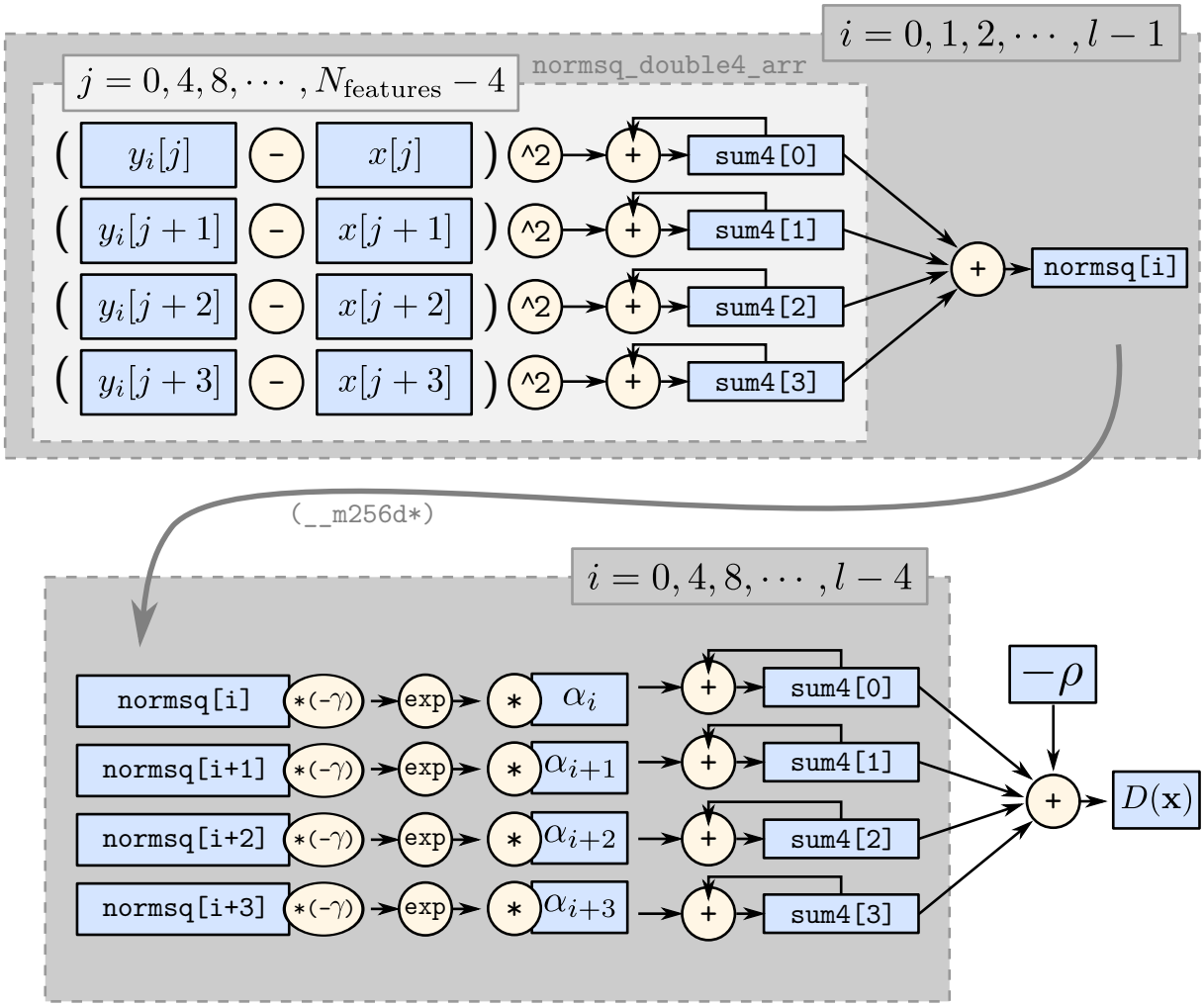

Figura 4.16- Otimização do classificador SVM com o uso de instruções vetoriais da família AVX. O cálculo pode ser realizado em duas etapas — primeiramente são obtidos todos os quadrados das distâncias entre vetores, e apenas em seguida é aplicado o núcleo RBF. Ambas as etapas podem ser paralelizadas em vetores de 4 elementos.

Fonte: Elaborada pelo autor.

4.8.1 Implementação da SVM com instruções vetoriais

Após treinado, um modelo SVM com núcleo RBF é composto pelos parâmetros γ (núcleo RBF), ρ (*bias*), por um conjunto de l vetores de suporte $\mathbf{y}_i$ e por um conjunto de l coeficientes α_i (pesos). Para classificar um vetor de características $\mathbf{x}$ a partir do modelo, calcula-se o valor de decisão $D(\mathbf{x})$:

$$D(\mathbf{x}) = -\rho + \sum_{i=0}^{l} \alpha_i \exp(-\gamma \|\mathbf{y}_i - \mathbf{x}\|^2) \tag{4.8.1}$$

Quando $D(\mathbf{x}) \geq 0$, classificamos $\mathbf{x}$ como produzido pelo peixe A. Caso contrário, classificamos como emitido pelo peixe B.

Para executar 4 operações em paralelo, utilizamos o tipo de dados `__m256d`, que representa, em linguagem C, o conteúdo de um registrador AVX com 4 elementos de precisão dupla.

O valor de l deve ser expandido, se necessário, para torná-lo um múltiplo de 4. Essa condição pode ser facilmente alcançada adicionando-se alguns elementos $\alpha_i = 0$ ao vetor de coeficientes, de forma a não alterar o resultado final do cálculo.

Primeiro, implementamos a função `normsq_double4_arr`, representada no quadro superior da Figura 4.16, que recebe os vetores $\mathbf{y}_i$ e $\mathbf{x}$, e retorna $\|\mathbf{y}_i - \mathbf{x}\|^2$. Fazemos uma conversão (*casting*) dos vetores de entrada para o tipo `__m256d*`, o que permite que eles sejam percorridos de 4 em 4 elementos. Para cada conjunto de 4 elementos, calculamos a diferença entre os vetores, quadramos essa diferença, e acumulamos o resultado em um registrador `sum4`. Ao final, a função retorna os 4 elementos de `sum4` somados entre si.

Chamamos essa função para $0 \leq i < l$, e armazenamos os resultados de $\|\mathbf{y}_i - \mathbf{x}\|^2$ em um vetor `normsq`, cujo endereço é alinhado em 32 bytes. Esse alinhamento é necessário para que possamos converter o vetor para `__m256d*`, mais uma vez permitindo o acesso simultâneo de 4 elementos. Da mesma forma, convertemos também para `__m256d*` o vetor que armazena os l valores de α_i.

Em seguida, como mostrado no quadro inferior da Figura 4.16, inicializamos com zeros um novo registrador `sum4`. Percorremos, simultaneamente, `normsq` e α. Multiplicamos cada conjunto de 4 elementos de `normsq` pelo escalar $-\gamma$, calculamos a exponencial dos 4 valores

simultaneamente, multiplicamos o vetor resultante pelo conjunto de 4 elementos de α, e acumulamos o resultado em `sum4`.

Para calcular a exponencial de 4 elementos simultaneamente, adaptamos uma implementação da biblioteca ROOT,[75] desenvolvida pelo CERN, para funcionar em linguagem C pura, facilitando assim sua compilação para o espaço de *kernel*. A rotina original era implementada em C++ empregando extensivamente o recurso de *templates* dessa linguagem.

Por fim, os 4 elementos de `sum4` são somados entre si, e o escalar ρ é subtraído do resultado, fornecendo o valor de decisão $D(\mathbf{x})$ do modelo SVM.

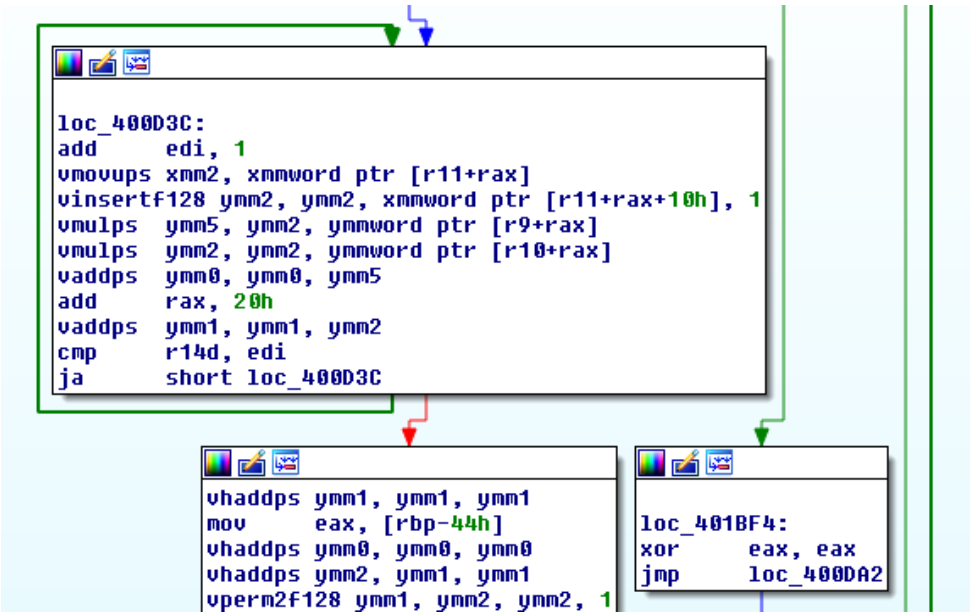

Figura 4.17- A transformada *wavelet* (DT-CWPT), apesar de não ter sido manualmente otimizada com instruções AVX, faz algum uso destas devido à vetorização automática do compilador GCC. As instruções `vmulps` e `vaddps` são pertencentes ao conjunto AVX e, respectivamente, multiplicam ou somam um vetor de 8 números de ponto flutuante de precisão simples.

Fonte: Elaborada pelo autor.

4.8.2 Ajustes finos na implementação de DT-CWPT

Como o cálculo dos vetores de características pela DT-CWPT era originalmente a etapa mais rápida dentre as 3 principais etapas do algoritmo de discriminação, realizamos apenas alguns ajustes finos para permitir que o próprio compilador (GCC) fosse capaz de inferir o uso de instruções AVX em algumas partes do código por meio do recurso de vetorização automática, como exemplifica a Figura 4.17.

O principal ajuste foi realizado na rotina de filtro de análise (`afb`), que acessava os elementos do vetor de entrada `in` da seguinte forma: `in[(ni + k) & n1]`. Assim, mesmo que o vetor `in` estivesse alinhado em 32 bytes, o compilador não era capaz de inferir acessos vetoriais devido ao operador AND bit-a-bit (&), utilizado para fazer *wrap-around* no vetor (voltar ao elemento de índice zero quando o elemento de índice `n` é atingido).

A solução foi, logo no início da rotina, pre-inicializar um vetor `extin` (alinhado em 32 bytes) com elementos `in[i & n1]`, variando `i` sobre toda a faixa acessada pela rotina ($0 \leq \mathtt{i} < \mathtt{off} + \mathtt{n} + \mathtt{ncoef}$). O tempo adicional necessário para essa cópia é compensado pelo tempo ganho empregando-se instruções vetoriais durante o cálculo, que ocorre em um laço de $(\mathtt{n}/2) \cdot \mathtt{ncoef}$ iterações.

4.9 Adaptação do RTAI para uso de instruções AVX

Quando uma interrupção é disparada, o processador pausa imediatamente o programa atualmente em execução e pula para uma rotina definida pelo sistema operacional. Nem o Linux nem o RTAI salvam, por padrão, os registradores de ponto flutuante antes de executar rotinas de interrupção. Isso significa que caso a rotina realize algum cálculo com ponto flutuante, modificando o estado desses registradores, qualquer programa em espaço de usuário que esteja realizando cálculos pode ter seus resultados corrompidos (devido aos valores intermediários contidos nos registradores serem alterados) ou mesmo travar.

O RTAI disponibiliza as macros `save_fpenv` e `restore_fpenv` para sanar esse problema em aplicativos de tempo real que necessitem fazer operações de ponto flutuante em espaço de kernel. Desta forma, a rotina desenvolvida pelo programador pode salvar o estado atual do programa que estava previamente em execução chamando `save_fpenv`, e restaurá-lo antes de retornar, chamando `restore_fpenv`.

No entanto, as macros disponibilizadas pelo RTAI atualmente utilizam a instrução `fxsave`, que salva apenas o estado utilizado por instruções de ponto flutuante simples (x87) ou de unidades vetoriais que já existem há mais tempo na arquitetura x86 (MMX, SSE e SSE2).

Como nossas implementações em software utilizam instruções AVX, adaptamos as macros e suas funções relacionadas para chamar a instrução `xsave`, como ilustra a Figura 4.18. Nossa

implementação foi validada deixando, por algumas horas, um processo executando em espaço de usuário que realizava diversas operações com instruções AVX e constantemente verificava se o resultado estava correto. Paralelamente a isso, um código que realizava as mesmas operações era executado em espaço de *kernel*, também verificando se os resultados estavam corretos, em resposta a uma interrupção constantemente disparada pelo nosso hardware.

```
#define TASK_FPENV(tsk)  (&(tsk)->thread.fpu.state->xsave)

static inline int __save_fpenv(struct xsave_struct __user *fx)
{
    int err;

    __asm__ __volatile__("1: .byte " REX_PREFIX "0x0f,0xae,0x27\n"
                         "2:\n"
                         ".section .fixup,\"ax\"\n"
                         "3:  movl $-1,%[err]\n"
                         "    jmp  2b\n"
                         ".previous\n"
                         ".section __ex_table,\"a\"\n"
                         _ASM_ALIGN "\n"
                         _ASM_PTR "1b,3b\n"
                         ".previous"
                         : [err] "=r" (err)
                         : "D" (fx), "a" (-1), "d" (-1), "" (0)
                         : "memory");

    return err;
}
```

```
static inline int __restore_fpenv(struct xsave_struct *fx)
{
    int err;

    asm volatile("1: .byte " REX_PREFIX "0x0f,0xae,0x2f\n\t"
                 "2:\n"
                 ".section .fixup,\"ax\"\n"
                 "3:  movl $-1,%[err]\n"
                 "    jmp  2b\n"
                 ".previous\n"
                 _ASM_EXTABLE(1b, 3b)
                 : [err] "=r" (err)
                 : "D" (fx), "m" (*fx), "a" (-1), "d" (-1), "" (0)
                 : "memory");

    return err;
}

#define save_fpenv(fpenv)     do { \
    __save_fpenv(&(fpenv).xsave); } while (0)
#define restore_fpenv(fpenv)  do { \
    __restore_fpenv(&(fpenv).xsave); } while (0)
```

Figura 4.18- Funções e macros modificadas no arquivo `rtai_fpu_64.h` para possibilitar o uso de instruções AVX em espaço de *kernel* nas rotinas de tempo real. Substituímos a instrução `fxsave`, que salva apenas o estado utilizado por instruções x87, MMX, SSE e SSE2, pela `xsave`, capaz de salvar o estado relacionado a quaisquer extensões suportadas pelo processador.

Fonte: Elaborada pelo autor.

Capítulo 5

Resultados

5.1 Acurácia de classificação

Primeiramente, avaliamos a capacidade do modelo SVM de, utilizado sozinho, classificar corretamente janelas contendo uma única DOE, comparando sua acurácia com o método de correlação cruzada (seção 3.1). Na Tabela 5.1, apresentamos a acurácia de ambos os classificadores para 6 díades diferentes de *Gymnotus* sp. Para a SVM, realizamos dois tipos diferentes de validação: simples e cruzada de 10 dobras (*10-fold cross validation*), ao passo que para o método de correlação cruzada, efetuamos apenas a validação simples.

Tabela 5.1- Acurácia da SVM, estimada por validação cruzada em 10 dobras e por validação simples, comparada com a acurácia do método de correlação cruzada, estimada por validação simples, em 6 díades diferentes.

Número da díade	Acurácia da SVM (validação cruzada 10-d.)	Acurácia da SVM (validação simples)	Acurácia da corr. cruz. (validação simples)
1	99,9828%	99,9778%	98,4660%
2	99,9998%	100,000%	99,4736%
3	99,9366%	99,9439%	96,4739%
4	99,9993%	99,9991%	99,8789%
5	99,9064%	99,9129%	91,6464%
6	99,9993%	99,9982%	99,2834%

Fonte: Elaborada pelo autor.

A validação simples consiste em classificar DOEs de origem conhecida pertencentes a um conjunto de teste escolhido aleatoriamente e disjunto dos conjuntos usados para treinamento (subseção 3.2.4). Já a validação cruzada de 10 dobras consiste em dividir a série de dados em

10 partes e treinar a SVM várias vezes, cada vez excluindo uma das partes, sempre medindo a acurácia do modelo resultante na classificação das 9 demais partes. O objetivo é obter uma boa estimativa de quão acurado será o modelo quando apresentado a um conjunto de dados estatisticamente independente daqueles utilizados no treinamento.

A acurácia da SVM estimada pela validação simples permaneceu, em geral, bastante próxima da acurácia estimada pela validação cruzada. A taxa de erros ($100\% -$ acurácia) obtida pela SVM foi, na maioria das vezes, 2 ordens de grandeza menor que a obtida pela correlação cruzada (díades 1, 3, 4 e 5), atingindo algumas vezes uma redução de 3 ordens de grandeza (díades 2 e 6). Tal fato atesta a superioridade do método descrito neste trabalho, diretamente relacionada à qualidade dos parâmetros e métodos computacionais escolhidos e da instrumentação eletrônica por nós desenvolvida.

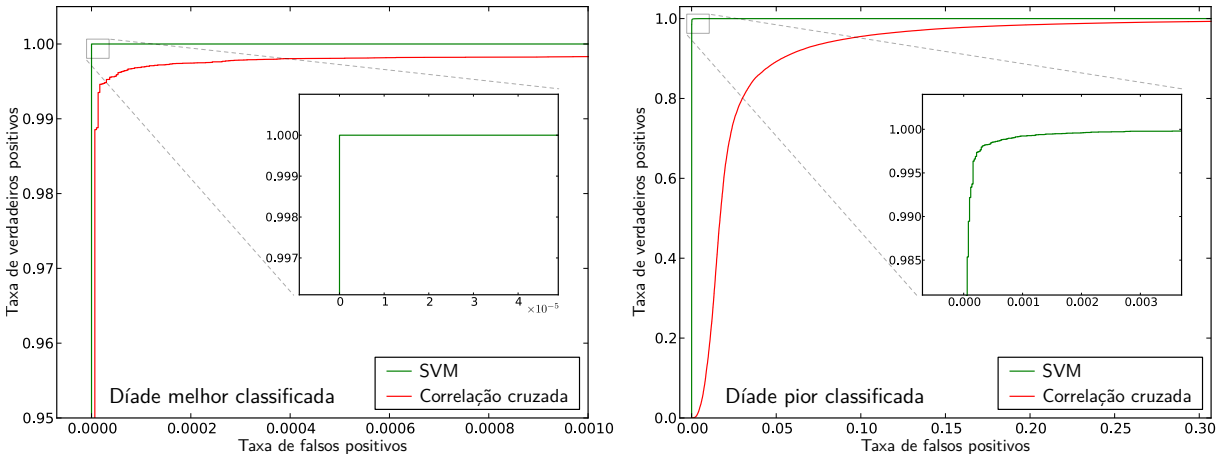

Figura 5.1- Curvas de característica de operação do receptor (COR) comparando os métodos de SVM e de correlação cruzada ao classificar janelas contendo um único DOE emitido por um indivíduo de um par (peixes A e B). A taxa de verdadeiros positivos denota a razão de DOEs emitidos pelo peixe A que foram identificados corretamente. De forma similar, a taxa de falsos positivos corresponde à razão dos DOEs emitidos pelo peixe B que foram identificados incorretamente. Os gráficos foram traçados para dois pares de peixes distintos — aqueles que obtiveram a melhor e a pior acurácia de classificação, respectivamente, dentre as 6 díades usadas nos experimentos.

Fonte: Elaborada pelo autor.

Na Figura 5.1, plotamos as curvas de característica de operação do receptor (COR) de ambos os classificadores para as díades nas quais ocorreu a melhor (díade 2) e a pior (díade 5) acurácia. As curvas COR mostram que nossa abordagem baseada em SVM é consistentemente superior ao método de correlação cruzada. Para a díade 5, na qual os pulsos de DOE emitidos por peixes diferentes possuíam praticamente a mesma duração característica, a SVM demonstra uma acurácia particularmente superior.

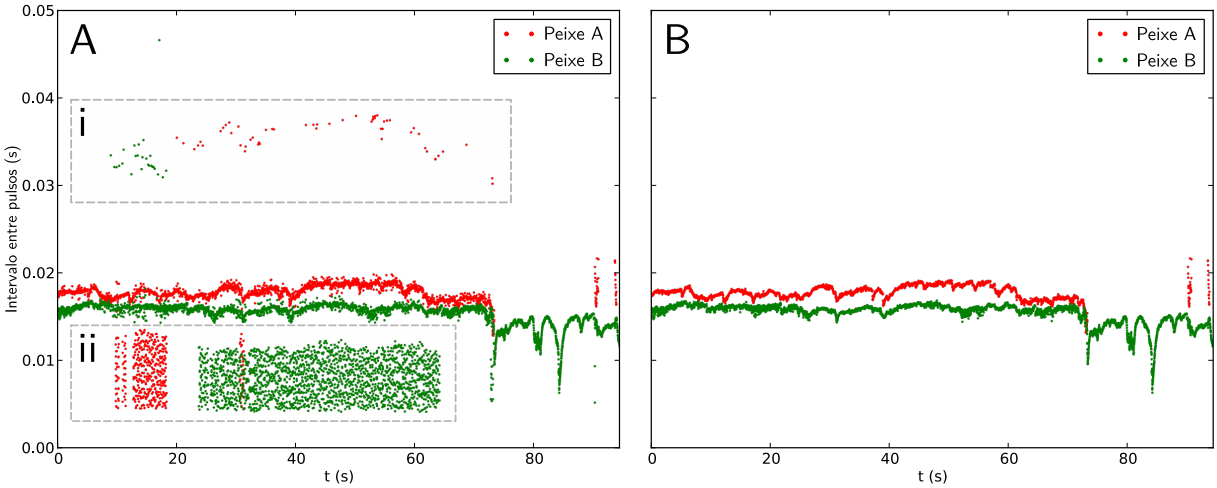

Figura 5.2- Gráficos de intervalo entre pulsos (IEP) apresentando dados provenientes de dois algoritmos distintos de discriminação, aos quais foram fornecidos os mesmos dados, coletados de um par de peixes que nadavam livremente. A sub-figura (A) mostra resultados de uma versão de nosso algoritmo[76] que usava somente a SVM (sem se aproveitar da propriedade de continuidade das formas de onda), e portanto apresentava dificuldades quando ambos os peixes emitiam DOEs quase ao mesmo tempo ou quando algum canal de medida saturava. Os erros de discriminação podem ser imediatamente observados no gráfico: a região (i) contém DOEs não-detectadas, e está localizada aproximadamente no dobro do IEP médio; a região (ii) contém falsos positivos, e apresenta IEPs abaixo do patamar médio. A sub-figura (B) mostra resultados de nosso algoritmo implementado por completo, como descrito neste trabalho. Nenhum erro pode ser apontado neste gráfico de IEP.

Fonte: Elaborada pelo autor.

Analisamos, então, dados coletados durante o procedimento de validação descrito na seção 2.3. Comparando medidas efetuadas diretamente por eletrodos anexados aos peixes com os resultados do algoritmo de discriminação, encontramos uma única DOE não-detectada dentre $7{,}4 \times 10^5$ pulsos, resultando em uma taxa de erros da ordem de duas partes por milhão de DOEs. Para a taxa de disparo de dois *Gymnotus* sp. típicos, isso representa um intervalo médio entre erros de uma hora de aquisição de dados. Essa taxa de erros é cerca de 4 ordens de grandeza inferior à que seria obtida pelo método de correlação cruzada.

O fato do sistema como um todo ser capaz de obter uma taxa de erros inferior à do classificador SVM isolado (Tabela 5.1) deve-se ao fato de que o resultado da classificação de vários canais de medida independentes é combinado para chegar ao resultado final. Além disso, a condição de continuidade (subseção 3.2.5) provê meios para classificar DOEs (tomando por base os disparos anteriores) quando a probabilidade de acerto da SVM é considerada baixa devido ao valor de decisão cair próximo à margem do classificador (subseção 3.2.4).

Finalmente, observamos gráficos de intervalo entre pulsos (IEP) construídos com resultados

do algoritmo de discriminação aplicado a dados coletados com ambos os peixes de uma díade nadando livremente, sem quaisquer eletrodos diretamente anexados. Como os indivíduos não foram restringidos de forma alguma ao longo do teste, podemos dizer que este foi conduzido da forma mais próxima possível à situação natural dos animais. Mesmo que medidas diretas não estejam disponíveis neste tipo de experimento, uma forma de validação ainda pode ser efetuada, pois é esperado que as curvas de IEP de um peixe sejam contínuas por partes. Um indivíduo pode parar de emitir DOEs por um pequeno período de tempo, porém enquanto ele ainda está emitindo DOEs, a taxa de disparos, e portanto também o IEP, variam de forma suave. Como mostrado na Figura 5.2, uma DOE perdida poderia ser imediatamente observada em um gráfico de IEP, pois apareceria como um ponto localizado no dobro do IEP do patamar médio. Similarmente, quaisquer DOEs detectadas pelo algoritmo que de fato não existissem apareceriam abaixo do patamar.

5.2 Desempenho da implementação de tempo real

Nesta seção comparamos, no quesito desempenho, cada etapa do algoritmo de discriminação que foi otimizada ou implementada em hardware (capítulo 4) com as etapas correspondentes da implementação *off-line* (capítulo 3).

Todas as medidas de tempo de execução de software foram aferidas em um computador com processador Intel Core i3-2120 de 3,30 GHz, por meio de um laço de 1000 execuções seguidas. Medimos o tempo total do laço com a rotina `omp_get_wtime` da biblioteca do OpenMP, e dividimos esse tempo por 1000 para obter o tempo de uma única execução. Repetimos esse procedimento de medida 64 vezes, e apresentamos o desvio padrão na forma de barras de erro nos gráficos. Já as medidas de tempo de execução de algoritmos implementados em hardware dedicado não são acompanhadas de barras de erro, pois possuem um desvio desprezível (da ordem do *jitter* do *clock*, que é de picossegundos).

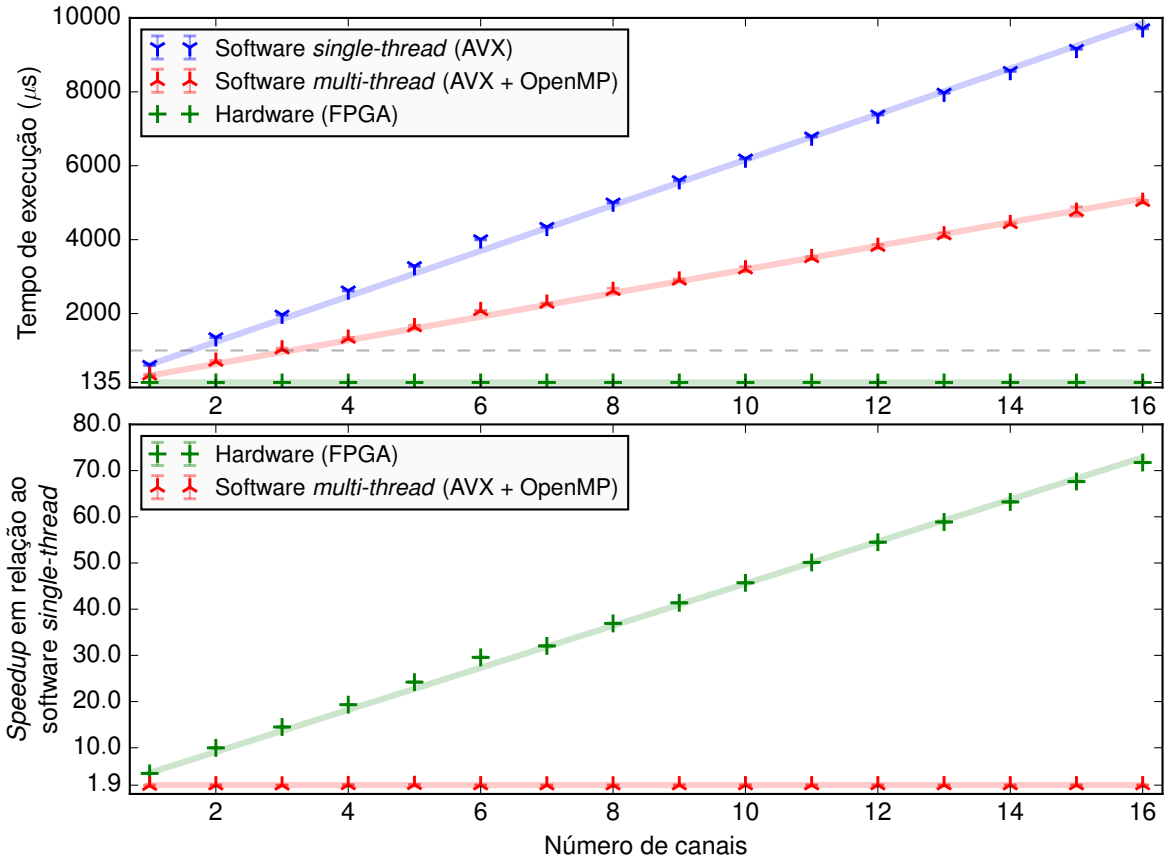

Figura 5.3- Tempo de execução e *speedup* de diferentes implementações do módulo de sobreposição de disparos, variando-se o número de canais e mantendo-se fixa uma janela de 128 amostras. Enquanto o hardware executa em tempo praticamente constante (135 μs), o tempo de execução da implementação em software cresce linearmente, e em torno de 3 canais já não é mais aceitável para uma implementação de tempo real. O *speedup* do hardware é praticamente linear com o número de canais, enquanto que uma implementação em software *multi-thread* alcança um *speedup* aproximadamente constante de 1,9$\times$.

Fonte: Elaborada pelo autor.

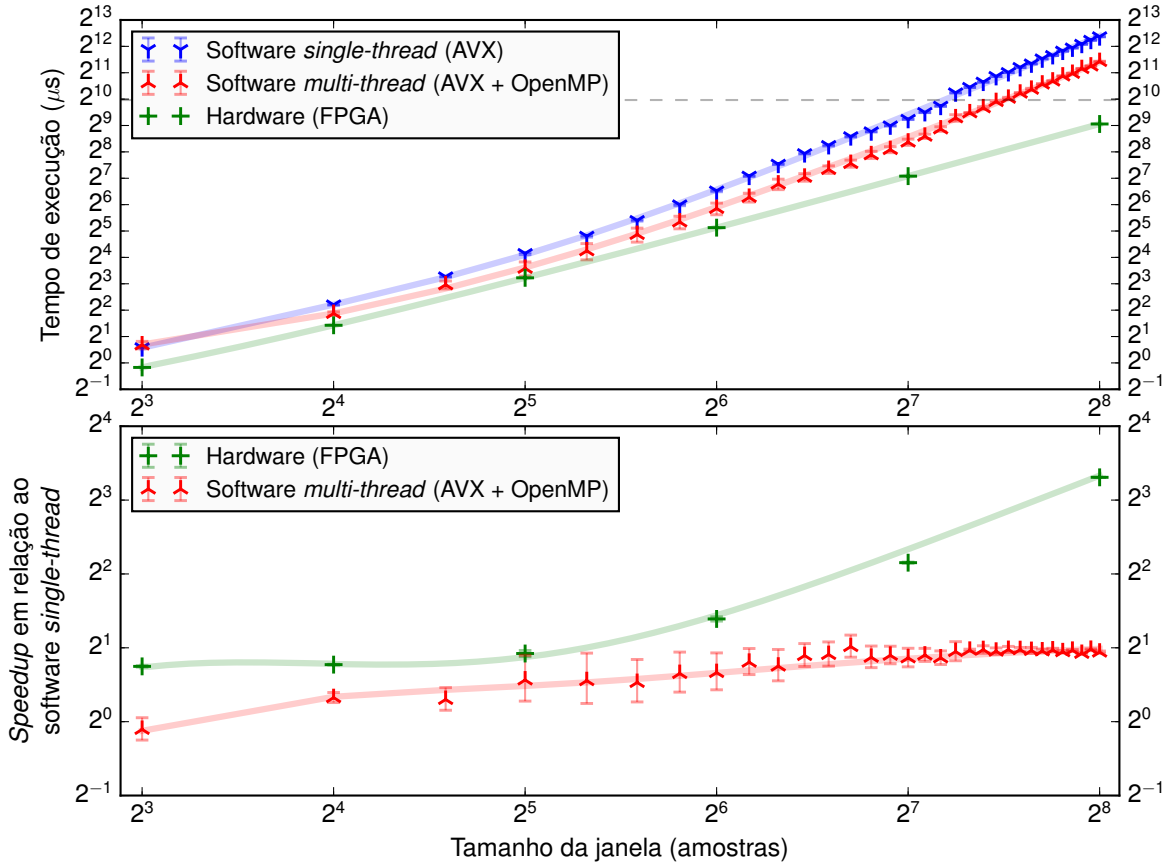

Figura 5.4- Tempo de execução e *speedup* de diferentes implementações do módulo de sobreposição de disparos, variando-se o número de amostras da janela e processando-se sempre um único canal. Os tempos de execução do hardware e do software foram ajustados, respectivamente, a polinômios do segundo e do terceiro grau, por se tratarem de algoritmos $\Theta(L_{\mathrm{w}}^2)$ e $\Theta(L_{\mathrm{w}}^3)$. O software *multi-thread* é mais lento que o *single-thread* para 8 amostras, mas torna-se gradativamente mais rápido aumentando-se a janela, até saturar com *speedup* praticamente ideal ($2\times$ em um computador de 2 núcleos). A implementação em hardware é sempre mais rápida, e tende a um *speedup* linear para janelas grandes ($>$64 amostras).

Fonte: Elaborada pelo autor.

5.2.1 Módulo de sobreposição de disparos

O módulo de sobreposição de disparos em hardware (implementado na seção 4.6) foi simulado com o Bluesim (simulador com precisão de ciclos pertencente ao pacote do Bluespec) para diversos valores de tamanho de janela (L_{w}) e de número de canais (N_{ch}). Constatamos por meio dessas simulações que todas as medidas de tempo de execução (em número de ciclos) satisfaziam à equação 5.2.1.

$$\text{Ciclos}(L_{\mathsf{w}}, N_{\mathsf{ch}}) = L_{\mathsf{w}}^2 + 4L_{\mathsf{w}} + \lceil \log_2 L_{\mathsf{w}} \rceil + \lceil \log_2 N_{\mathsf{ch}} \rceil + 12 \tag{5.2.1}$$

Esse comportamento era esperado: o termo quadrático em L_{w} é o tamanho do espaço de busca, o termo linear em L_{w} está relacionado ao teste de janelas contendo um único peixe, e os termos logarítmicos em L_{w} e em N_{ch} devem-se ao tempo de propagação dos dados na árvore de somadores. O termo constante (12) contempla a latência resultante da inserção de filas (FIFOs) entre os diversos módulos do hardware, utilizadas para desacoplá-los e reduzir a ocorrência de grandes caminhos combinacionais.

A implementação em software desta etapa do algoritmo, que havia sido avaliada de forma preliminar na seção 3.3, foi agora otimizada manualmente com instruções AVX, considerando uma janela de tamanho fixo e múltiplo de 8. Obtemos assim uma implementação com maior desempenho e com tempo de execução mais determinístico, visando realizar a comparação mais justa possível com a implementação em hardware maciçamente paralelo.

A Figura 5.3 compara as implementações variando o número de canais (N_{ch}) e fixando o tamanho da janela em $L_{\mathsf{w}} = 128$ amostras, mesmo valor que é utilizado em nossa implementação final. Como os cálculos associados a todos os canais são efetuados em paralelo, o hardware executa no tempo praticamente constante de 135 μs — esse tempo varia somente de 32 ns ($\log_2 16 = 4$ ciclos) de uma ponta a outra do gráfico, mudança que pode ser considerada desprezível dentro da escala de tempo associada a este problema. No entanto, o software executa em um tempo que cresce linearmente com o número de canais, e que com mais de um único canal já estoura o requisito de 1 ms. Para este tamanho de janela, é possível paralelizar com bastante sucesso o software, obtendo-se um *speedup* próximo do ideal (1,9$\times$) para um processador de 2 núcleos, mas ainda assim é possível processar no máximo 2 canais antes de

atingir o limite de tempo, isso sem considerar as demais etapas necessárias para executar por completo o algoritmo de discriminação.

Na Figura 5.4, variamos o tamanho da janela (L_{w}) mantendo fixo $N_{\mathsf{ch}} = 1$. Enquanto o tempo de execução das implementações em software cresce com uma dependência cúbica, o hardware dedicado executa em um tempo predominantemente quadrático (equação 5.2.1). A maior janela que a implementação em software paralelo é capaz de processar em um tempo inferior a 1 ms é de $L_{\mathsf{w}} = 176$ amostras, ao passo que o hardware é capaz de cumprir os requisitos de tempo real mesmo para janelas de $L_{\mathsf{w}} = 256$ amostras, executando em 533 μs.

5.2.2 SVM com instruções AVX

A Figura 5.5 compara o tempo de execução das bibliotecas `libsvm` e `libsvm-dense`, ambas versão 3.20, com a implementação otimizada com instruções AVX proposta na subseção 4.8.1. Ambas as bibliotecas podem ser utilizadas tanto em modo *single* como *multi-thread*, no qual acrescentam-se diretivas OpenMP para distribuir o cálculo do núcleo RBF entre os diferentes núcleos do processador, ao passo que a implementação proposta é executada somente em modo *single-thread*. Em cada uma das 64 medidas de tempo realizadas para cada ponto apresentado no gráfico, foi utilizado um novo modelo SVM, gerado aleatoriamente.

As medidas demonstram que, como seria esperado, a biblioteca `libsvm-dense` apresenta sempre desempenho superior à `libsvm` para nossa aplicação, que não utiliza vetores de características esparsos. Pode-se notar também que, para modelos com poucos vetores de suporte ($N_{\mathsf{SV}} \leq 64$), as implementações *multi-thread* são mais lentas em média que as implementações *single-thread*, ou seja, não compensam nem para processamento *off-line*. Esses modelos SVM de baixa complexidade são de fato observados na prática, como revela a Tabela 5.2.

A implementação proposta é capaz de classificar 11 vetores de características diferentes (um para cada canal de nosso maior aquário) em um tempo de $(110{,}0 \pm 0{,}6)$ ns por vetor de suporte e, apesar de *single-thread*, é mais rápida que todas as outras implementações de classificador SVM que foram testadas, além de apresentar uma variabilidade muito menor do tempo de execução, como evidenciado pelas barras de erro do gráfico.

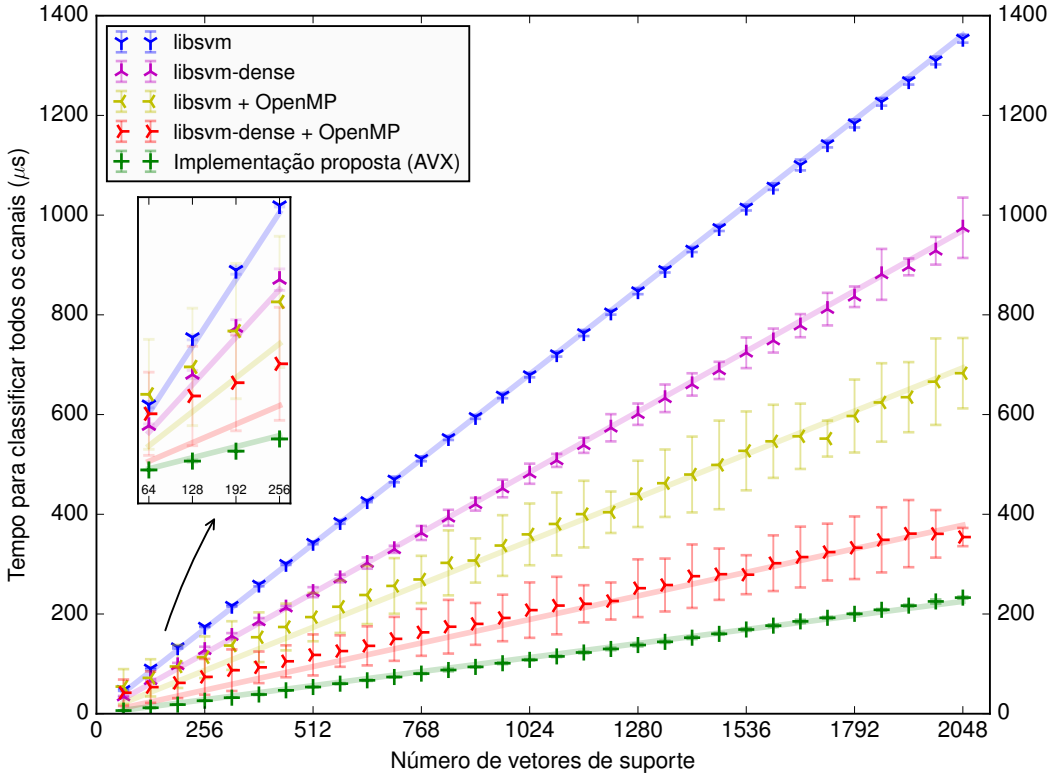

Figura 5.5- Tempo de execução de diferentes implementações de SVM (apenas classificação), variando-se o número de vetores de suporte e processando-se sempre 11 canais. Para nossa aplicação, a biblioteca `libsvm-dense` é mais rápida que a `libsvm` padrão, entretanto nossa implementação com instruções AVX é sempre superior a ambas as bibliotecas (tanto em suas versões *single* como *multi-thread*), além de apresentar uma dispersão muito menor do tempo de execução.

Fonte: Elaborada pelo autor.

Tabela 5.2- Quantidade de vetores de suporte da SVM treinada com 6 díades diferentes.

Número da díade	Quantidade de vetores de suporte da SVM
1	593
2	60
3	1102
4	46
5	1106
6	53

Fonte: Elaborada pelo autor.

5.2.3 Cálculo dos vetores de características

A implementação anterior aos ajustes discutidos na subseção 4.8.2 executava em aproximadamente 340 μs para 11 canais de entrada. Após os ajustes, o tempo foi reduzido para $(293 \pm 4)\,\mu$s, um ganho de cerca de 1,2$\times$.

5.3 Consumo de recursos de hardware do FPGA

As Figuras 5.6 e 5.7 apresentam, respectivamente, o consumo de blocos lógicos e de blocos de memória do sistema sintetizado em um FPGA da linha Altera Stratix IV, modelo `EP4SGX230KF40C2`, com o Quartus II versão 15.0.2 Build 153. Variamos o número de canais do conversor AD habilitados para uso pelo discriminador, além de sintetizarmos versões com e sem o filtro decimador (seção 4.1) na entrada do módulo de sobreposição de disparos. Com o decimador, esse módulo trabalha com janelas de 128 amostras, ao passo que sem o decimador, as janelas passam a ser de 256 amostras.

A quantidade de blocos DSP de 18-bit ocupados, 8 dos 1288 disponíveis no FPGA, não é mostrada nas figuras pois permanece constante. Esses blocos são utilizados pelos multiplicadores pertencentes ao filtro IIR e à lógica que calcula o módulo da transformada de Hilbert, ambos internos ao módulo delimitador de janelas (seção 4.4).

A quantidade de blocos lógicos utilizados aumenta de forma aproximadamente linear com o número de canais habilitados. Entretanto, ao tentarmos sintetizar uma versão do sistema com 9 canais habilitados e janela de 256 amostas que, extrapolando o ajuste linear, deveria ocupar 8126 LABs (89% do FPGA), ocorre uma descontinuidade nesse comportamento: o *fitter* do Quartus II aborta o processo de síntese prematuramente, informando que seriam necessários 9557 LABs (105% do FPGA) para implementar o projeto. Esse fenômeno provavelmente deve-se a uma tentativa do Quartus II de replicar lógica para minimizar o caminho crítico combinacional quando os recursos do FPGA estão chegando próximos ao limite.

A quantidade de blocos de memória varia de forma um pouco mais complicada com o número de canais (N_{ch}), pois a largura de alguns barramentos de memória do projeto depende de $\lceil \log_2 N_{\mathsf{ch}} \rceil$, fazendo com que, além do termo linear com N_{ch}, exista um termo proporcional

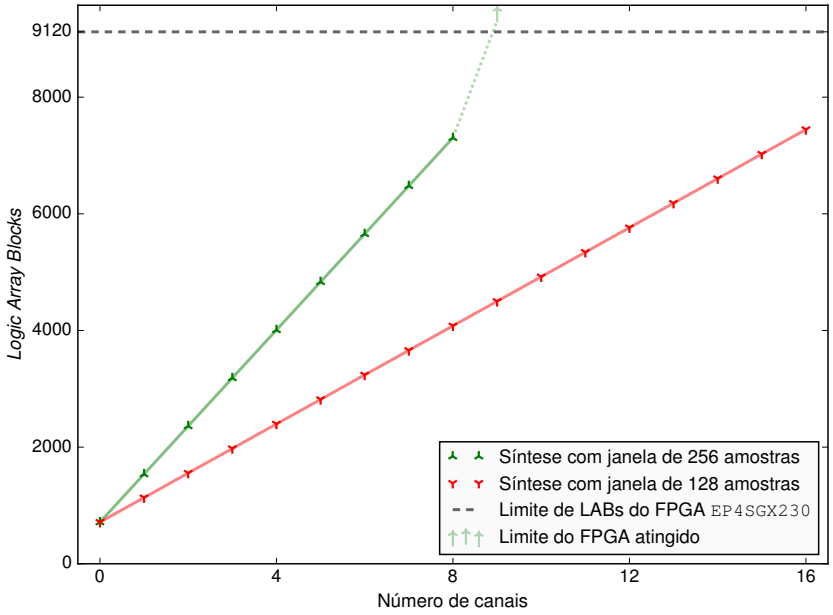

Figura 5.6- A quantidade de blocos lógicos utilizados no FPGA cresce de forma aproximadamente linear com o número de canais de entrada habilitados, exceto quando a ocupação se aproxima de 8126 LABs (89% dos blocos lógicos). Quando o módulo de sobreposição de disparos é configurado para utilizar uma janela de 256 amostras, o EP4SGX230 comporta no máximo 8 canais. Ativando-se o filtro decimador na entrada do módulo (que reduz a janela para 128 amostras), é possível habilitar todos os 16 canais oferecidos pelo conversor AD.

Fonte: Elaborada pelo autor.

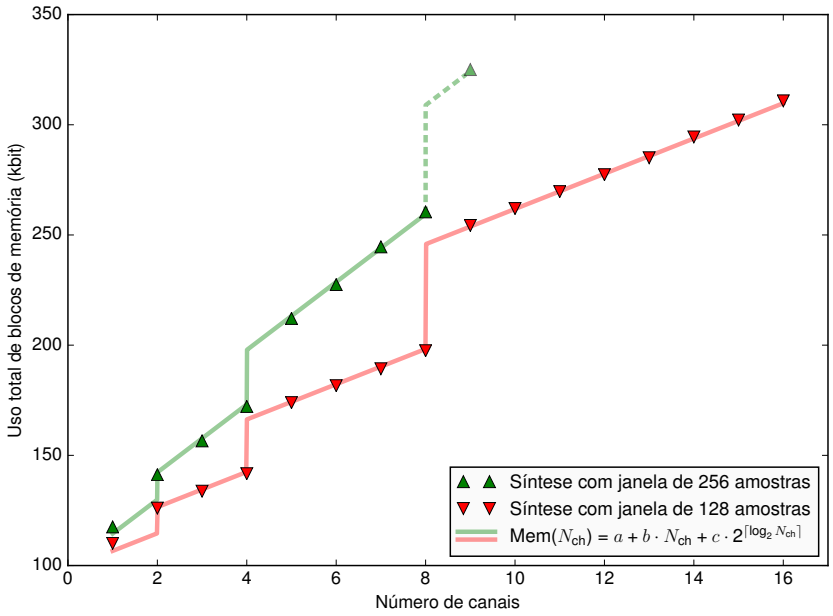

Figura 5.7- A quantidade de blocos de memória utilizados no FPGA aumenta proporcionalmente ao número de canais e à menor potência de 2 maior ou igual a este, permanecendo porém sempre muito menor que o limite de 14 625 kbit oferecido pelo EP4SGX230.

Fonte: Elaborada pelo autor.

à menor potência de 2 maior ou igual a N_{ch}. No entanto, a ocupação de blocos de memória é sempre muito baixa, não superando 2% do FPGA em nenhuma das versões sintetizadas.

Removendo-se o módulo de sobreposição de disparos, o projeto de hardware ainda pode ser utilizado como um sistema de aquisição genérico, com opção de realizar uma aquisição contínua (subseção 4.3.3) ou de detectar o instante de ocorrência (*timestamp*) de DOEs de peixes elétricos ou de potenciais de ação de neurônios (*spikes*), transferindo para o computador somente janelas ao redor das DOEs ou *spikes* (seção 4.5). Nesse caso, o sistema ocupa e 714 LABs (8% do FPGA) e 100 512 bits em blocos de memória ($<1\%$ do FPGA).

A frequência máxima para a linha de *clock* principal do sistema, estimada pelo modelo lento a 900 mV e 85°C, variou de 138,64 MHz (para a versão sem módulo de sobreposição de disparos) até 125,69 MHz (para 16 canais, com janela de 128 amostras). A queda da frequência máxima com o aumento do número de canais habilitados não foi monotônica, indicando que existe uma variabilidade aleatória de acordo com as decisões tomadas pelo *fitter* do Quartus II. O requisito mínimo para a linha de *clock* principal deste projeto é de 125 MHz, ou seja, os requisitos de tempo foram cumpridos por todas as versões sintetizadas.

5.4 Latência total

A Figura 5.8 ilustra a montagem experimental destinada a aferir a latência total de nosso discriminador de DOEs. Foi escolhida uma díade cujo treinamento do classificador resultou em um modelo SVM contendo um número relativamente grande de vetores de suporte ($N_{\text{SV}} = 1613$).

Os três sinais diferentes monitorados pelo osciloscópio são detalhados na Figura 5.9: o primeiro é o valor negado da soma de todos os 11 canais do osciloscópio, calculado por um circuito analógico, que permite observar diretamente o instante de ocorrência da DOE com relação às respostas do discriminador; o segundo é um sinal digital que é posto em nível lógico alto pela rotina de interrupção do software de tempo real assim que esta começa a executar, e colocado de volta em nível baixo logo antes da rotina finalizar; o terceiro é um sinal produzido por um módulo de hardware que é programado pela rotina para disparar 2 ms após o pico da soma de transformadas de Hilbert de toda DOE que for detectada como produzida pelo peixe A (primeiro peixe da díade). Esse último sinal é utilizado como *trigger* do osciloscópio,

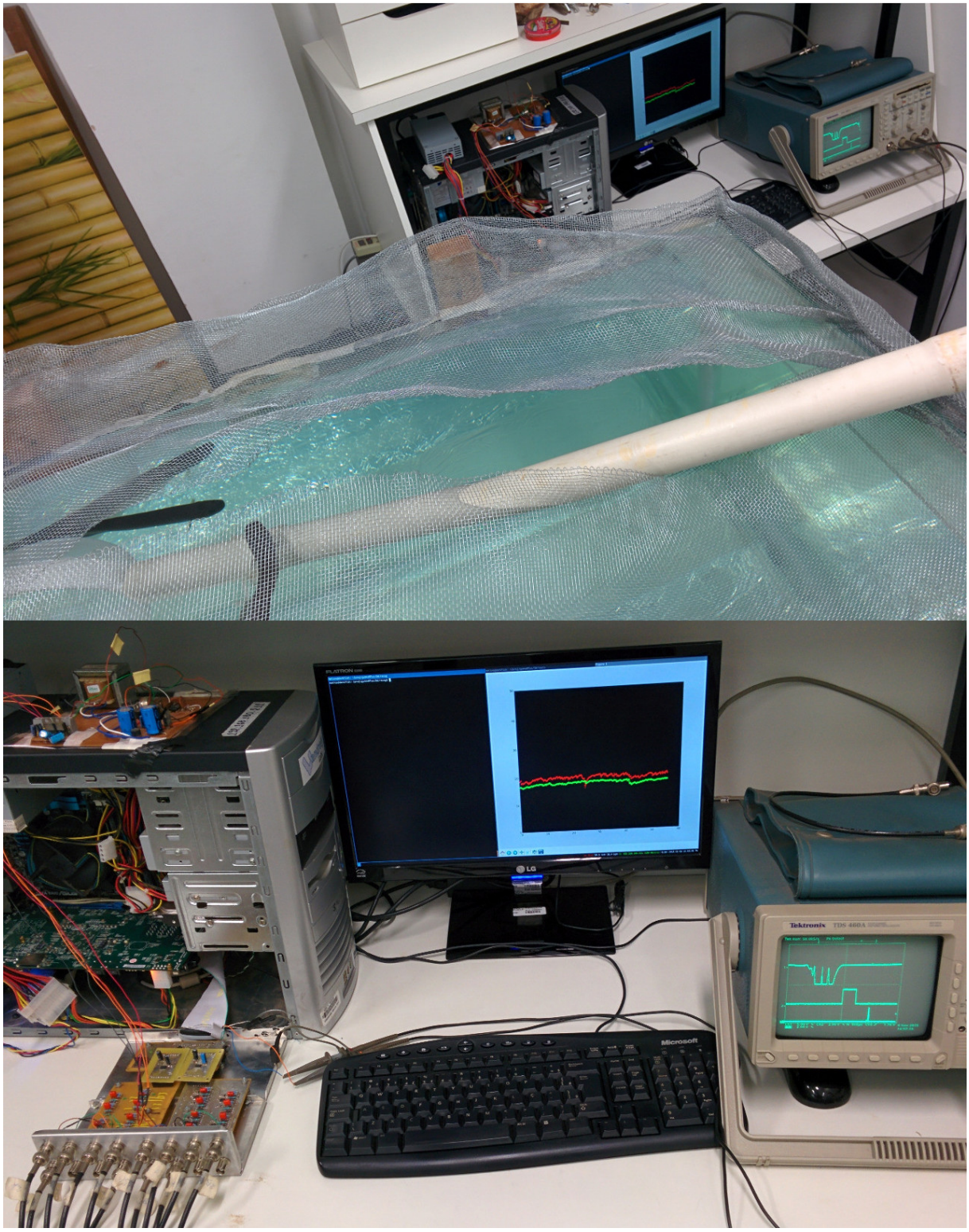

Figura 5.8- Experimento de discriminação em tempo real das DOEs de uma díade. A tela do computador exibe um gráfico de IEP de ambos os peixes que é atualizado *on-line*, ou seja, ao mesmo tempo em que as DOEs são discriminadas. Mesmo com a carga do sistema inerente a disponibilizar uma interface gráfica interativa, o discriminador é capaz de atender aos requisitos de tempo real.

Fonte: Elaborada pelo autor.

Figura 5.9- Tela do osciloscópio mostrando a captura de duas DOEs quase consecutivas. O canal 1 (em verde) corresponde ao valor negado da soma dos quadrados dos sinais de todos os 11 canais do aquário. O canal 2 (em vermelho) apresenta nível lógico alto durante todo o período em que a DOE está sendo processada pela rotina de tempo real no RTOS. O canal 3 (em azul) é programado para disparar 2 ms após o ponto de referência (pico da soma de transformadas de Hilbert) de toda DOE que for detectada como proveniente do peixe A.

Fonte: Elaborada pelo autor.

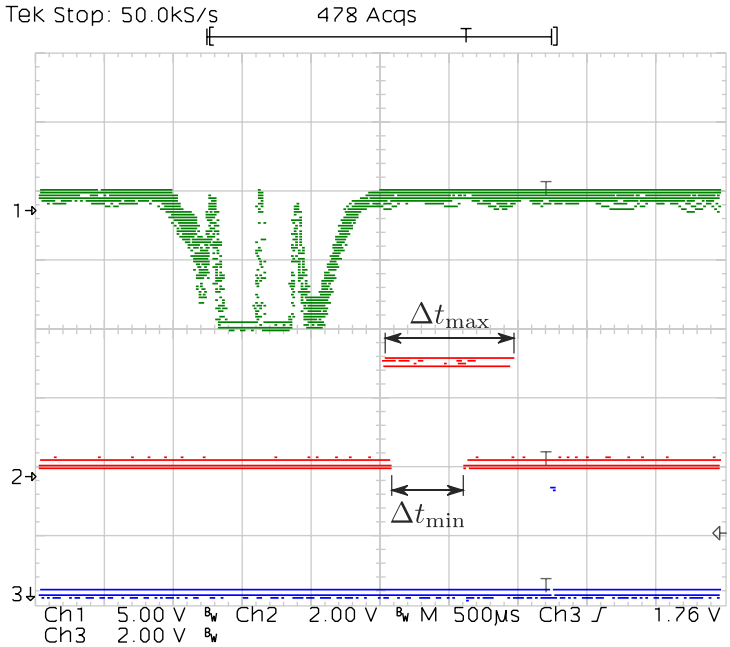

Figura 5.10- A latência mínima ($\Delta t_{\text{min}} \approx 500\mu$s) e a máxima ($\Delta t_{\text{max}} \approx 950\mu$s) do discriminador são aferidas ativando o modo de persistência do osciloscópio e definindo o canal 3 (em azul) como *trigger* a fim de alinhar os sinais observados com um ponto de referência aproximadamente fixo.

Fonte: Elaborada pelo autor.

pois produz um pulso a uma distância constante de um ponto de referência aproximadamente fixo dentro de toda DOE produzida por um dos peixes.

Ao mesmo tempo em que um software executado no Linux atualiza constantemente (*on-line*) um gráfico da IEP de cada um dos peixes, a rotina de interrupção do discriminador é executada no domínio de tempo real (RTAI) sem deixar de atender ao requisito de latência máxima de 1 ms, como demonstra a Figura 5.10.

De acordo com o ajuste linear apresentado na subseção 5.2.2, o tempo de execução da etapa de classificação (SVM) estimado para o valor de N_{SV} desta díade é de $(177 \pm 1)\,\mu$s. Somando ao tempo necessário para a outra etapa de maior peso computacional executada pelo software de tempo real, que é o cálculo dos vetores de características (subseção 5.2.3), estimamos um tempo total de $(470 \pm 5)\,\mu$s. Essa estimativa é consistente com a latência mínima ($\Delta t_{\mathsf{min}} \approx 500\,\mu$s) aferida com o auxílio do osciloscópio.

A latência máxima ($\Delta t_{\mathsf{max}} \approx 950\,\mu$s), apesar de atingir quase o dobro da latência mínima, atende plenamente a nossos requisitos de tempo real. Entretanto, acreditamos que ainda existe espaço para melhorias por meio de alguns ajustes finos, por exemplo utilizar mecanismos disponibilizados por versões mais recentes do RTAI para isolar o domínio de tempo real em um único núcleo do processador, reduzindo a poluição do *cache* L1 e outros fatores que possam aumentar a variabilidade da latência.

Capítulo 6

Sugestões para trabalhos futuros

6.1 Utilização do sistema

6.1.1 Experimentos com realimentação elétrica

A literatura conta com diversos trabalhos envolvendo a estimulação de peixes elétricos pulsadores com DOEs artificiais. No entanto, até o presente momento esses trabalhos eram restritos à experimentação com um único animal, já que anteriormente a esta tese não existia nenhum sistema capaz de discriminar DOEs de díades em tempo real.

Em particular, experimentos envolvendo a geração de estímulos síncronos às DOEs são de grande interesse, e vem sendo propostos desde os primeiros estudos com esses animais. Westby[3] relatou que *Gymnotus* sp. só reage (com alterações em sua taxa de disparo) nos casos em que o atraso de pulsos de estímulo produzidos sincronicamente esteja fora de um intervalo de 0,6 ms a 12,2 ms após a detecção de cada DOE, a menos que os pulsos sejam gerados com amplitude alta o suficiente para simular a existência de outro peixe muito próximo (4 cm ou menos de distância) ao animal estudado. Trabalhos recentes[30] têm realizado experimentos similares com peixes pulsadores de outras espécies, como *Gnathonemus petersii*. Temos conhecimento também de experimentos atualmente em andamento com *Gymnotus* sp. no grupo de pesquisa do professor Angel A. Caputi, no Instituto de Investigaciones Biológicas Clemente Estable, envolvendo a produção de estímulos síncronos às DOEs de um peixe e a investigação da relação entre suas respostas e medidas intracranianas realizadas no núcleo

magnocelular.

O discriminador desenvolvido no presente trabalho torna viável a execução de diversas variações inéditas desses experimentos utilizando díades. Por exemplo, poderíamos gerar pulsos de estímulo com um atraso fixo com relação à emissão das DOEs de apenas um dos peixes da díade, dentro do intervalo descrito como "cego" por Westby.[3] Neste caso, seria esperado que apenas o outro peixe reagisse ao estímulo. Em uma versão um pouco mais sofisticada do experimento, poderíamos avaliar se esse tipo de estímulo é capaz de alterar a relação de dominância e submissão entre peixes. Um modelo poderia tentar prever o próximo instante de disparo do peixe dominante com base na continuidade dos IEPs anteriormente observados, e gerar um pulso sobreposto às DOEs deste peixe sempre que caíssem no intervalo tido como "cego" para o peixe submisso. A intenção seria causar um distúrbio para o peixe dominante na interpretação de seu próprio disparo, de uma forma que afetasse o mínimo possível o peixe submisso.

No entanto, nosso sistema não se restringe à geração de estímulos síncronos, sendo capaz de alimentar qualquer modelo de estimulação em *closed-loop* que receba como entrada os instantes de disparo das DOEs de cada um dos peixes, por exemplo aplicando conceitos de teoria da informação, codificação e linguagem.

6.1.2 Experimentos com realimentação luminosa

Experimentos preliminares do Grupo de Neurocomputación Biológica da Universidad Autónoma de Madrid em colaboração com o pesquisador Lirio O. B. de Almeida, do Laboratório de Neurobiofísica do IFSC–USP, demonstraram que *Gnathonemus petersii* pode apresentar variações em sua resposta elétrica quando apresentado a um estímulo luminoso síncrono com suas DOEs. Ainda não houve tentativa de reproduzir esses experimentos com *Gymnotus* sp., porém caso resultados similares sejam observados, o sistema introduzido por esta tese possibilitaria a realização do experimento com uma díade, permitindo que o estímulo luminoso fosse sincronizado às DOEs de somente um dos peixes colocados no aquário.

6.2 Implementação de variações do sistema

6.2.1 Implementação em kits FPGA de baixo custo

No projeto aqui apresentado, foi utilizado o kit Terasic DE4-230, que no momento da escrita é vendido a US$ 2995. Uma proposta interessante seria implementar uma versão de mais baixo custo do discriminador, utilizando por exemplo um kit Terasic DE1-SoC, baseado em um FPGA da família Altera Cyclone V, disponível ao custo de somente US$ 175 para usuários acadêmicos.

Uma vantagem do DE1-SoC com relação ao kit DE4 utilizado neste trabalho é o fato do DE1-SoC possuir embarcado na placa o conversor AD7928,[77] de 8 canais, com resolução de 12 bits e taxa de amostragem de até 1 MS/s dividida entre os canais (125 kS/s por canal). Desta forma, um sistema que trabalhasse com aquários de até 8 canais poderia ser implementado sem necessitar das placas externas com conversor MAX1280[43] construídas como parte desta tese. Seria necessário apenas o *front-end* analógico (subseção 4.3.1) para condicionar os sinais na faixa aceita pelo AD7928, que é a mesma do MAX1280 (0 a 2,5 V).

O DE1-SoC não pode ser conectado ao barramento PCI Express de um computador, porém o modelo de FPGA adotado (`5CSEMA5F31C6`) possui, além dos blocos lógicos reprogramáveis, um processador ARM Cortex-A9 de 2 núcleos fabricado na mesma pastilha de silício, que pode ser utilizado para co-projeto de hardware/software. Entretanto, seria necessário avaliar se esse processador possui desempenho suficiente para executar as etapas do algoritmo que foram implementadas em software. Uma alternativa seria manter os módulos de software em arquitetura PC, utilizando rede Gigabit Ethernet em vez de PCI Express para comunicação entre o FPGA e o computador. Essa comunicação poderia ser implementada com baixa latência utilizado o RTnet,[78] um *framework* de rede para RTAI que cumpre requisitos de tempo real, sendo capaz de atingir um *round trip time* da ordem de 60 μs.[79]

Para ter uma noção da viabilidade de implementar os módulos de hardware no FPGA do kit DE1-SoC, sintetizamos o módulo principal em Bluespec, variando o número de canais habilitados até atingir o máximo possível (6 canais, com janelas de 128 amostras no módulo de sobreposição de disparos). Foram utilizados 28 546 ALMs (89% dos blocos lógicos do

FPGA), 136 880 bits em blocos de memória (3% do FPGA) e 5 blocos DSP (6% do FPGA). A frequência máxima de *clock* estimada pelo modelo lento a 1100 mV e 85°C foi de 107,1 MHz. Como o tempo de execução é inversamente proporcional ao *clock*, arredondando para uma frequência de 100 MHz, o módulo de sobreposição de disparos seria capaz de executar em 169 μs em vez dos 135 μs obtidos na Stratix IV. Ainda assim, esse tempo é curto o bastante para cumprir, com folga, nossos requisitos de tempo real.

Uma alternativa para obter um sistema capaz de ser conectado a um aquário com mais de 6 canais seria realizar um compromisso entre a utilização de blocos lógicos e o tempo de execução. Para isso, seria necessário modificar a arquitetura de forma que apenas parte dos canais fosse processada em paralelo, guardando um resultado intermediário em uma memória RAM de alta velocidade a cada ciclo de relógio. Essa RAM precisaria armazenar uma matriz de $128 \times 128 \times 24$ bits, ocupando assim apenas cerca de 9% dos blocos de memória do FPGA. Desta forma, seria possível dobrar o número de canais para 12, dobrando também o tempo de execução para 338 μs, valor que ainda atende aos requisitos de tempo real.

6.2.2 Implementação *standalone* em FPGA

Caso o algoritmo de discriminação fosse implementado inteiramente em hardware, o sistema resultante seria autocontido (*standalone*), portátil, e apresentaria um *jitter* muito inferior ao obtido em um co-projeto com RTOS. Como o projeto atual necessita de uma pequena quantidade de blocos multiplicadores, que são primordiais para implementações eficientes de classificadores SVM e de transformadas Fourier ou *wavelet* em hardware, provavelmente existe espaço suficiente para a implementação desses módulos, dependendo do número de canais desejado e da consequente ocupação de blocos lógicos pelo módulo de sobreposição de disparos. Os tópicos a seguir trazem uma breve revisão da literatura relacionada à implementação de cada um desses módulos.

Classificador SVM em hardware

Anguita et al.[80] trazem um dos primeiros estudos sobre o assunto. Os ensaios conduzidos por esses autores demonstram que o uso de aritmética de ponto fixo não costuma degradar signifi-

cativamente a precisão da SVM com relação a implementações utilizando ponto flutuante. Em alguns casos, o uso de ponto fixo pode até ser benéfico para a acurácia do algoritmo, evitando o aparecimento de problemas de sobreajustamento (*overfitting*). O FPGA utilizado por esses autores é cerca de sete vezes menor que nosso EP4SGX230, e a arquitetura sugerida no artigo ocupa apenas 6% da mesma. Todavia, o *clock* máximo alcançado é somente de cerca de 20 MHz, muito menor que o obtido por trabalhos posteriores, o que contudo não significa que a arquitetura proposta não seja eficiente. Talvez esse fato se dê devido a diferenças entre tecnologias de fabricação de FPGAs.

Irick et al.[81] focam no uso de um classificador SVM previamente treinado, em vez de tentar implementar o algoritmo de treinamento em FPGA. Essa abordagem também seria adequada para nós, visto que nossa SVM não precisa ser treinada em tempo real. O modelo de FPGA utilizado não é especificado no artigo, afirma-se porém ter sido possível operar a uma frequência de 100 MHz.

Cadambi et al.[82] utilizam uma abordagem de co-projeto de hardware/software, na qual o FPGA atua como um acelerador cuja execução é comandada por um computador, por meio de um duto PCI Express. Os procedimentos de treino e classificação tornam-se 18,2 e 21,6 vezes mais rápidos, respectivamente, com o uso desse acelerador. A arquitetura proposta ocupa até 37549 elementos lógicos e 128 blocos multiplicadores de um FPGA Xilinx Virtex-5, que seriam equivalentes a 20900 dos elementos (9,2%) e 10% dos multiplicadores de nosso EP4SGX230.[83] O *clock* obtido é de cerca de 140 MHz.

Papadonikolakis et al.[84] propõem um mecanismo de cascata para otimizar a relação entre a proporção de elementos lógicos e de blocos multiplicadores utilizados em um FPGA, promovendo a maior ocupação possível do chip. Os trabalhos anteriores a esse empregam uma tabela pré-computada (e armazenada em blocos de memória RAM internos) para o cálculo da RBF. Entretanto, nessa arquitetura, é utilizada uma abordagem completamente diferente: a cascata fornece um resultado de ponto fixo, que é convertido para ponto flutuante antes de passar para o bloco que calcula o núcleo RBF, sem o uso de uma tabela. A arquitetura proposta pelos autores preenche quase completamente os elementos lógicos de um EP3SE260. Nosso EP4SGX230 fornece um número ligeiramente menor de elementos lógicos, porém possui

um número maior de blocos multiplicadores (1288 contra 768 do FPGA adotado no artigo). Os resultados do trabalho são excelentes — um ganho de desempenho de 2 a 3 ordens de grandeza é obtido com relação a implementações em software, e a abordagem é sete vezes mais rápida que trabalhos similares utilizando FPGAs ou GPUs.

Cálculo dos vetores de características em hardware

Para calcular nossos vetores de características, utilizamos principalmente a transformada complexa de dupla árvore de pacote *wavelet*. Apesar de não conhecermos nenhum trabalho a respeito da implementação desse tipo de transformada *wavelet* em FPGAs, outras famílias são largamente abordadas na literatura.[85–91] Porém, como toda transformada *wavelet* é composta essencialmente pelas mesmas operações básicas de aplicação de um conjunto de filtros de resposta ao impulso finita, podemos adaptar facilmente as mesmas arquiteturas atualmente descritas na literatura.

A transformada de Fourier, que também pode ser utilizada para calcular elementos de nossos vetores de características, está associada a uma extensa literatura. Porém, a implementação de Man Cheuk Ng et al.[63,64] é particularmente interessante, por ser parametrizável e escrita em Bluespec, a mesma linguagem de descrição de hardware adotada neste trabalho.

Capítulo 7

Conclusões

Apresentamos um método capaz de reconhecer o indivíduo que emitiu cada descarga de órgão elétrico (DOE) em experimentos realizados com pares de peixes *Gymnotus* sp. nadando livremente. Os dados obtidos são úteis para analisar e estudar protocolos de comunicação empregados pelos animais em uma variedade de situação interessantes, como acasalamento, estabelecimento de relações de dominância e disputa territorial, além de serem importantes para trazer mais luz a problemas de base, tais como eficiência e redundância da codificação de sinais de comunicação, respostas para evitar ruído[92] e mecanismos de multiplexação de canais que podem estar presentes nesses animais.

A montagem experimental necessária para replicar a implementação mais simples de nosso método, que discrimina séries de dados previamente gravadas (de forma *off-line*), é de fácil acesso a qualquer laboratório de eletrofisiologia — um arranjo de eletrodos fixo, circuitos convencionais com amplificadores operacionais para condicionar os sinais, e um sistema de aquisição de dados. Os eletrodos podem ser afixados em um aquário ou montados em uma estrutura que pode ser colocada em um viveiro. Ao contrário de procedimentos adotados em estudos anteriores, não são necessárias câmeras, permitindo assim que experimentos sejam facilmente realizados em águas túrbidas, que são um habitat comum para esses animais.[93] Além disso, somos capazes de processar corretamente uma grande quantidade de dados coletados, o que é essencial para a obtenção de resultados mais confiáveis quando são aplicadas ferramentas de teoria da informação e abordagens estatísticas para analisar os sinais de comunicação.

Experimentos conduzidos utilizando *Gymnotus* sp. produziram resultados de discriminação

excelentes, com taxas de erro (aferidas comparando-se o resultado do algoritmo com medidas diretas) da ordem de algumas partes por milhão. Portanto, acreditamos que esse método possa também ser aplicado a outras espécies de peixes elétricos pulsadores, restando apenas verificar se essas espécies possuem características individuais distinguíveis pelo nosso classificador baseado em máquinas de vetores de suporte (SVM).

Construímos uma implementação de tempo real do nosso método, capaz de discriminar DOEs de peixes elétricos com uma latência máxima de 1 ms após a ocorrência de cada pulso. Validamos essa implementação com aquários de até 12 eletrodos (11 canais amplificados diferencialmente), porém demonstramos que ela seria capaz de escalar bem para aquários com até 16 canais de saída. Medimos o instante de ocorrência (*timestamp*) de cada DOE em hardware, portanto a resolução temporal da medida não é afetada pela variabilidade no tempo de execução (*jitter*) do software de tempo real.

Aferimos o instante de ocorrência, sempre que possível, com base no pico da soma de módulos da transformada de Hilbert de todos os canais do aquário, método que é conhecido por fornecer resultados com precisão temporal da mesma ordem que nosso período de amostragem (20 μs).[8] A resolução temporal cai pela metade (40 μs) apenas quando são detectadas janelas de sinal contendo disparos sobrepostos de ambos os peixes, caso no qual o instante de ocorrência é aferido por um módulo de hardware que realiza tentativas de sobreposição dos disparos anteriormente observados.

A instrumentação desenvolvida para a implementação de tempo real apresenta um elevado grau de sofisticação mas, ainda assim, é replicável, pelo fato de adotar um kit de FPGA facilmente encontrado no mercado (Terasic DE4-230), e devido a todas as placas de circuito customizadas que conectamos a esse FPGA serem passíveis de fabricação por métodos de fácil acesso, como transferência térmica de toner.

Referências

1 BENNETT, M. V.; GRUNDFEST, H. Electrophysiology of electric organ in *Gymnotus carapo*. *J. Gen. Physiol.*, v. 42, n. 5, p. 1067–1104, 1959. doi: 10.1085/jgp.42.5.1067.

2 CAPUTI, A. A.; CASTELLÓ, M. E.; AGUILERA, P.; TRUJILLO-CENÓZ, O. Electrolocation and electrocommunication in pulse gymnotids: signal carriers, pre-receptor mechanisms and the electrosensory mosaic. *J. Physiol.-Paris*, v. 96, n. 5–6, p. 493–505, 2002. doi: 10.1016/S0928-4257(03)00005-6.

3 WESTBY, G. W. M. Has the latency dependent response of *Gymnotus carapo* to discharge-triggered stimuli a bearing on electric fish communication? *J. Comp. Physiol.*, v. 96, n. 4, p. 307–341, 1975. doi: 10.1007/BF00619223.

4 WONG, R. Y.; HOPKINS, C. D. Electrical and behavioral courtship displays in the mormyrid fish *Brienomyrus brachyistius*. *J. Exp. Biol.*, v. 210, p. 2244–2252, 2007. doi: 10.1242/jeb.003509.

5 GEBHARDT, K.; BÖHME, M.; VON DER EMDE, G. Electrocommunication behaviour during social interactions in two species of pulse-type weakly electric fishes (Mormyridae). *J. Fish Biol.*, v. 81, n. 7, p. 2235–2254, 2012. doi: 10.1111/j.1095-8649.2012.03448.x.

6 MATIAS, P.; GUARIENTO, R. T.; DE ALMEIDA, L. O. B.; SLAETS, J. F. W. Modular acquisition and stimulation system for timestamp-driven neuroscience experiments. In: INTERNATIONAL SYMPOSIUM ON APPLIED RECONFIGURABLE COMPUTING, 11., 2015, Bochum, DE. *Proceedings...* Cham, CH: Springer, c2015. p. 339–348 (Lecture Notes in Computer Science, v. 9040). doi: 10.1007/978-3-319-16214-0_29.

7 WESTBY, G. W. M. Comparative studies of the aggressive behaviour of two gymnotid electric fish (*Gymnotus carapo* and *Hypopomus artedi*). *Anim. Behav.*, v. 23, p. 192–213, 1975. doi: 10.1016/0003-3472(75)90065-2.

8 JUN, J. J.; LONGTIN, A.; MALER, L. Precision measurement of electric organ discharge timing from freely moving weakly electric fish. *J. Neurophysiol.*, v. 107, n. 7, p. 1996–2007, 2012. doi: 10.1152/jn.00757.2011.

9 FORLIM, C. G.; PINTO, R. D. Automatic realistic real time stimulation/recording in weakly electric fish: long time behavior characterization in freely swimming fish and stimuli discrimination. *PLoS ONE*, v. 9, n. 1, p. e84885, 2014. doi: 10.1371/journal.pone.0084885.

10 GROSS, G. W.; RIESKE, E.; KREUTZBERG, G. W.; MEYER, A. A new fixed-array multi-microelectrode system designed for long-term monitoring of extracellular single unit neuronal activity in vitro. *Neurosci. Lett.*, v. 6, n. 2–3, p. 101–105, 1977. doi: 10.1016/0304-3940(77)90003-9.

11 KIPKE, D. R.; VETTER, R. J.; WILLIAMS, J. C.; HETKE, J. F. Silicon-substrate intracortical microelectrode arrays for long-term recording of neuronal spike activity in cerebral cortex. *IEEE T. Neur. Sys. Reh.*, v. 11, n. 2, p. 151–155, 2003. doi: 10.1109/TNSRE.2003.814443.

12 SNIDER, R. K.; BONDS, A. B. Classification of non-stationary neural signals. *J. Neurosci. Meth.*, v. 84, n. 1–2, p. 155–166, 1998. doi: 10.1016/S0165-0270(98)00110-1.

13 BATISTA, G.; ZUBIZARRETA, L.; PERRONE, R.; SILVA, A. Non-sex-biased dominance in a sexually monomorphic electric fish: fight structure and submissive electric signalling. *Ethology*, v. 118, n. 4, p. 398–410, 2012. doi: 10.1111/j.1439-0310.2012.02022.x.

14 PERRONE, R.; MACADAR, O.; SILVA, A. Social electric signals in freely moving dyads of *Brachyhypopomus pinnicaudatus*. *J. Comp. Physiol. A*, v. 195, n. 5, p. 501–514, 2009. doi: 10.1007/s00359-009-0427-6.

15 ARNEGARD, M. E.; CARLSON, B. A. Electric organ discharge patterns during group hunting by a mormyrid fish. *P. Roy. Soc. B*–biol. sci., v. 272, n. 1570, p. 1305–1314, 2005. doi: 10.1098/rspb.2005.3101.

16 MCGREGOR, P. K.; WESTBY, G. W. M. Discrimination of individually characteristic electric organ discharges by a weakly electric fish. *Anim. Behav.*, v. 43, n. 6, p. 977–986, 1992. doi: 10.1016/S0003-3472(06)80011-4.

17 LEWICKI, M. S. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network*–comp. neural, v. 9, n. 4, p. R53–R78, 1998. doi: 10.1088/0954-898X_9_4_001.

18 GALLANT, J. R.; ARNEGARD, M. E.; SULLIVAN, J. P.; CARLSON, B. A.; HOPKINS, C. D. Signal variation and its morphological correlates in *Paramormyrops kingsleyae* provide insight into the evolution of electrogenic signal diversity in mormyrid electric fish. *J. Comp. Physiol. A*, v. 197, n. 8, p. 799–817, 2011. doi: 10.1007/s00359-011-0643-8.

19 CRAMPTON, W. G. R.; HOPKINS, C. D. Nesting and paternal care in the weakly electric fish *Gymnotus* (Gymnotiformes: Gymnotidae) with descriptions of larval and adult electric organ discharges of two species. *Copeia*, v. 2005, n. 1, p. 48–60, 2005. doi: 10.1643/CI-04-056R1.

20 HOPKINS, C. D. Temporal structure of non-propagated electric communication signals. *Brain Behav. Evol.*, v. 28, n. 1–3, p. 43–59, 1986. doi: 10.1159/000118691.

21 STRONG, S. P.; KÖBERLE, R.; DE RUYTER VAN STEVENINCK, R. R.; BIALEK, W. Entropy and information in neural spike trains. *Phys. Rev. Lett.*, v. 80, p. 197–200, 1998. doi: 10.1103/PhysRevLett.80.197.

22 NEMENMAN, I.; LEWEN, G. D.; BIALEK, W.; DE RUYTER VAN STEVENINCK, R. R. Neural coding of natural stimuli: information at sub-millisecond resolution. *PLoS Comput. Biol.*, v. 4, n. 3, p. e1000025, 2008. doi: 10.1371/journal.pcbi.1000025.

23 CORTES, C.; VAPNIK, V. Support-vector networks. *Mach. Learn.*, v. 20, n. 3, p. 273–297, 1995. doi: 10.1007/BF00994018.

24 BAYRAM, I.; SELESNICK, I. W. On the dual-tree complex wavelet packet and M-band transforms. *IEEE Trans. Sig. Proc.*, v. 56, n. 6, p. 2298–2310, 2008. doi: 10.1109/TSP.2007.916129.

25 WEICKERT, T.; BENJAMINSEN, C.; KIENCKE, U. Analytic wavelet packets: combining the dual-tree approach with wavelet packets for signal analysis and filtering. *IEEE Trans. Sig. Proc.*, v. 57, n. 2, p. 493–502, 2009. doi: 10.1109/TSP.2008.2007922.

26 MUÑIZ, C.; RODRÍGUEZ, F.; VARONA, P. RTBiomanager: a software platform to expand the applications of real-time technology in neuroscience. *BMC Neurosci.*, v. 10, n. 1, p. 49, 2009. doi: 10.1186/1471-2202-10-S1-P49.

27 DE ALMEIDA, L. O. B.; MATIAS, P.; GUARIENTO, R. T. An embedded system for real-time feedback neuroscience experiments. In: BRAZILIAN SYMPOSIUM ON COMPUTING SYSTEMS ENGINEERING, 4., 2014, Manaus. *Proceedings...* Manaus: Instituto de Ciências Exatas, Universidade Federal do Amazonas, c2014. doi: 10.13140/RG.2.1.4077.7769.

28 SHARP, A. A.; O'NEIL, M. B.; ABBOTT, L. F.; MARDER, E. The dynamic clamp: artificial conductances in biological neurons. *Trends Neurosci.*, v. 16, n. 10, p. 389–394, 1993. doi: 10.1016/0166-2236(93)90004-6.

29 CHAMORRO, P.; MUÑIZ, C.; LEVI, R.; ARROYO, D.; RODRÍGUEZ, F. B.; VARONA, P. Generalization of the dynamic clamp concept in neurophysiology and behavior. *PLoS ONE*, v. 7, n. 7, p. e40887, 2012. doi: 10.1371/journal.pone.0040887.

30 FORLIM, C. G.; PINTO, R. D.; VARONA, P.; RODRÍGUEZ, F. B. Delay-dependent response in weakly electric fish under closed-loop pulse stimulation. *PloS ONE*, v. 10, n. 10, p. e0141007, 2015. doi: 10.1371/journal.pone.0141007.

31 DE MICHELL, G.; GUPTA, R. K. Hardware/software co-design. *P. IEEE*, v. 85, n. 3, p. 349–365, 1997. doi: 10.1109/5.558708.

32 NOGUEIRA, J.; CASTELLÓ, M. E.; CAPUTI, A. A. The role of single spiking spherical neurons in a fast sensory pathway. *J. Exp. Biol.*, v. 209, n. 6, p. 1122–1134, 2006. doi: 10.1242/jeb.02080.

33 LOMONT, C. *Introduction to Intel Advanced Vector Extensions*. Santa Clara, CA, US: Intel Corporation, 2011. Disponível em: <https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions>. Acesso em: 18 nov. 2015.

34 MANTEGAZZA, P.; DOZIO, E. L.; PAPACHARALAMBOUS, S. RTAI: real time application interface. *Linux Journal*, v. 72, p. A10, 2000.

35 NIKHIL, R. Bluespec System Verilog: efficient, correct RTL from high level specifications. In: ACM AND IEEE INTERNATIONAL CONFERENCE ON FORMAL METHODS AND MODELS FOR CO-DESIGN, 2., 2014, San Diego, CA, US. *Proceedings...* Piscataway, NJ, US: IEEE Press, c2004. p. 69–70. doi: 10.1109/MEMCOD.2004.1459818.

36 ROSENBAND, D. L.; ARVIND. Hardware synthesis from guarded atomic actions with performance specifications. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2005, San Jose, CA, US. *Proceedings...* Piscataway, NJ, US: IEEE Press, c2005. p. 784–791. doi: 10.1109/ICCAD.2005.1560170.

37 MAUCH, S.; REGER, J.; REINLEIN, C.; APPELFELDER, M.; GOY, M.; BECKERT, E.; TÜNNERMANN, A. FPGA-accelerated adaptive optics wavefront control. In: MEMS ADAPTIVE OPTICS, 8., 2014, San Francisco, CA, US. *Proceedings...* Bellingham, WA, US: SPIE, c2014. p. 897802:1–12 (Proceedings of SPIE, v. 8978). doi: 10.1117/12.2038910.

38 CHEN, Y.; SUN, K.; WANG, T.; WEI, H. Development and implementation of a real time embedded control system for machine tools. In: IEEE/ASME INTERNATIONAL CONFERENCE ON MECHATRONIC AND EMBEDDED SYSTEMS AND APPLICATIONS, 2., 2006, Beijing, CN. *Proceedings...* Pitscataway, NJ, US: IEEE Press, c2006. p. 1–5. doi: 10.1109/MESA.2006.296946.

39 ZHANG, W. J.; DUN, Y.; SUN, Y.; YU, L.; SHI, W.; ZHANG, W.; SHI, M. High productivity reconfigurable digital control system and its application on magnetic bearing. In: IEEE CONFERENCE ON INDUSTRIAL ELECTRONICS AND APPLICATIONS, 4., 2009, Xian, CN. *Proceedings...* Pitscataway, NJ, US: IEEE Press, c2009. p. 1844–1850. doi: 10.1109/ICIEA.2009.5138517.

40 WHITHAM, J.; AUDSLEY, N. C.; DAVIS, R. I. Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system. *ACM Trans. Embed. Comput. Syst.*, v. 13, n. 4s, p. 120:1–25, 2014. doi: 10.1145/2523070.

41 LYN, K. G. *Classification of and resilience to cyber-attacks on cyber-physical systems*. 2015. p. 86. (Masters Thesis) – Georgia Institute of Technology, Atlanta, GA, US, 2015. Disponível em: <http://hdl.handle.net/1853/53926>. Acesso em: 15 nov. 2015.

42 TEXAS Instruments. *LM108/LM208/LM308 operational amplifiers*. Dallas, TX, US, 2013. Disponível em: <http://www.ti.com/lit/ds/symlink/lm208-n.pdf>. Acesso em: 19 nov. 2015.

43 MAXIM Integrated. *400ksps/300ksps, single-supply, low-power, 8-channel, serial 12-bit ADCs with internal reference*: MAX1280/MAX1281. Sunnyvale, CA, US, 2010. Disponível em: <http://pdfserv.maximintegrated.com/en/ds/MAX1280-MAX1281.pdf>. Acesso em: 20 nov. 2015.

44 MATIAS, P.; SLAETS, J. F. W.; PINTO, R. D. Individual discrimination of freely swimming pulse-type electric fish from electrode array recordings. *Neurocomputing*, v. 153, p. 191–198, 2015. doi: 10.1016/j.neucom.2014.11.037.

45 MATIAS, P.; GUARIENTO, R. T. *gymnotools*. Software. Disponível em: <https://github.com/neurobiofisica/gymnotools>. Acesso em: 23 nov. 2015.

46 MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, v. 8, n. 1, p. 3–30, 1998. doi: 10.1145/272991.272995.

47 PAINTNER, S.; KRAMER, B. Electrosensory basis for individual recognition in a weakly electric, mormyrid fish, *Pollimyrus adspersus* (Günther, 1866). *Behav. Ecol. Sociobiol.*, v. 55, n. 2, p. 197–208, 2003. doi: 10.1007/s00265-003-0690-4.

48 KINGSBURY, N. Design of Q-shift complex wavelets for image processing using frequency domain energy minimization. In: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 10., 2003, Barcelona, ES. *Proceedings...* Tampere, FI: SuviSoft, c2003. p. 1013–1016. doi: 10.1109/ICIP.2003.1247137.

49 JOHN, G. H.; KOHAVI, R.; PFLEGER, K. Irrelevant features and the subset selection problem. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 11., 1994, New Brunswick, NJ, US. *Proceedings...* San Francisco, CA, US: Morgan Kaufmann, c1994. p. 121–129. doi: 10.1016/b978-1-55860-335-6.50023-4.

50 MEYER, D.; LEISCH, F.; HORNIK, K. The support vector machine under test. *Neurocomputing*, v. 55, n. 1–2, p. 169–186, 2003. doi: 10.1016/S0925-2312(03)00431-4.

51 CHANG, C.-C.; LIN, C.-J. LIBSVM: a library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, v. 2, n. 3, p. 27:1–27, 2011. doi: 10.1145/1961189.1961199.

52 HSU, C.-W.; CHANG, C.-C.; LIN, C.-J. *A practical guide to support vector classification*. Taipei, TW: Department of Computer Science, National Taiwan University, 2010. Technical report. Disponível em: <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>. Acesso em: 13 nov. 2015.

53 PLATT, J. C. Probabilities for SV machines. In: SMOLA, A. J.; BARTLETT, P. L.; SCHÖLKOPF, B.; SCHUURMANS, D. (Ed.) *Advances in large-margin classifiers*. Cambridge, MA, US: MIT Press, 1999. cap. 5, p. 61–74.

54 LIN, H.-T.; LIN, C.-J.; WENG, R. A note on Platt's probabilistic outputs for support vector machines. *Mach. Learn.*, v. 68, n. 3, p. 267–276, 2007. doi: 10.1007/s10994-007-5018-6.

55 DAGUM, L.; MENON, R. OpenMP: an industry standard API for shared-memory programming. *IEEE Comput. Sci. Eng.*, v. 5, n. 1, p. 46–55, 1998. doi: 10.1109/99.660313.

56 ARVIND; NIKHIL, R. S.; ROSENBAND, D. L.; DAVE, N. High-level synthesis: an essential ingredient for designing complex ASICs. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, 2004, San Jose, CA, US. *Proceedings...* Piscataway, NJ, US: IEEE Press, c2004. p. 775–782. doi: 10.1109/ICCAD.2004.1382681.

57 WADLER, P. Monads for functional programming. In: INTERNATIONAL SPRING SCHOOL ON ADVANCED FUNCTIONAL PROGRAMMING TECHNIQUES, 1., 1995, Bastad, SE. *Proceedings...* Berlin, DE: Springer, c1995. p. 24–52 (Lecture Notes in Computer Science, v. 925). doi: 10.1007/3-540-59451-5_2.

58 SHEN, X.; ARVIND. Design and verification of speculative processors. In: WORKSHOP ON FORMAL TECHNIQUES FOR HARDWARE AND HARDWARE-LIKE SYSTEMS, 1998, Marstrand, SE. *Proceedings...* Göteborg, SE: Computing Science Department, Chalmers University of Technology, c1998.

59 MATIAS, P. *gymnoRTus*. Software. Disponível em: <`https://github.com/thotypous/gymnoRTus`>. Acesso em: 23 nov. 2015.

60 MATIAS, P. *altsourceprobe*. Software. Disponível em: <`https://github.com/thotypous/altsourceprobe`>. Acesso em: 23 nov. 2015.

61 IEEE Computer Society. *IEEE standard for test access port and boundary-scan architecture*: IEEE 1149.1. New York, NY, US, 2013. doi: 10.1109/IEEESTD.2013.6515989.

62 MURRAY, K. E.; BETZ, V. Quantifying the cost and benefit of latency insensitive communication on FPGAs. In: ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS, 22., 2014, Monterey, California, USA. *Proceedings...* New York, NY, US: Association for Computing Machinery, c2014. p. 223–232. doi: 10.1145/2554688.2554786.

63 NG, M. C. *Airblue*: a highly-configurable FPGA-based platform for wireless network research. 2011. p. 164. Ph.D. Thesis – Massachusetts Institute of Technology, Cambridge, MA, US, 2011. Disponível em: <`http://hdl.handle.net/1721.1/66457`>. Acesso em: 13 nov. 2015.

64 FLEMING, K. E.; NG, M. C.; GROSS, S.; ARVIND. WiLIS: architectural modeling of wireless systems. In: IEEE INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE, 2011, Austin, TX, US. *Proceedings...* Piscataway, NJ, US: IEEE Press, c2011. p. 197–206. doi: 10.1109/ISPASS.2011.5762736.

65 MATIAS, P. *Measuring IRQ latency in RTAI*: PCIe vs LPT. São Carlos, 2015. Disponível em: <https://pmatias.me/2015-05/rtai-measuring-irq-latency>. Acesso em: 23 nov. 2015.

66 XENOMAI Project. *Analogy general presentation*. Paris, FR, 2014. Disponível em: <https://xenomai.org/analogy-general-presentation>. Acesso em: 29 nov. 2015.

67 NXP Semiconductors. *BAT54W series*: schottky barrier diodes. Eindhoven, NL, 2012. Disponível em: <http://www.nxp.com/documents/data_sheet/BAT54W_SER.pdf>. Acesso em: 29 nov. 2015.

68 TEXAS Instruments. *TLV271, TLV272, TLV274*: family of 500-μA/ch 3-MHz rail-to-rail output operational amplifiers. Dallas, TX, US, 2004. Disponível em: <http://www.ti.com/lit/ds/symlink/tlv271.pdf>. Acesso em: 29 nov. 2015.

69 TUMA, T.; BÜRMEN, Á. *Circuit simulation with SPICE OPUS*: theory and practice. Boston, MA, US: Birkhäuser, 2009.

70 BLUESPEC Inc. *Bluespec SystemVerilog reference guide*. Framingham, MA, US, 2012. Disponível em: <http://www.bluespec.com/forum/download.php?id=157>. Acesso em: 24 nov. 2015.

71 BALARIN, F.; PASSERONE, R. Functional verification methodology based on formal interface specification and transactor generation. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2006, Munich, DE. *Proceedings...* Los Alamitos, CA, US: IEEE Computer Society Press, c2006. p. 1013–1018. doi: 10.1109/date.2006.243899.

72 SHANKAR, S. S.; SHANKAR, J. S. Synthesizable verification IP to stress test system-on-chip emulation and prototyping platforms. In: INTERNATIONAL SYMPOSIUM ON INTEGRATED CIRCUITS, 13., 2011, Singapore. *Proceedings...* Piscataway, NJ, US: IEEE Press, c2011. p. 609–612. doi: 10.1109/ISICir.2011.6131936.

73 ACCELERA Systems Initiative. *Standard co-emulation modeling interface*: SCE-MI. Elk Grove, CA, US, 2015. Disponível em: <http://www.accellera.org/downloads/standards/sce-mi>. Acesso em: 13 nov. 2015.

74 ALTERA Corp. Design debugging using in-system sources and probes: verification. In: *Quartus II Handbook*: verification. San Jose, CA, US, 15th. ed., 2015. cap. 17, p. 381–394. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/qts_qii5v3.pdf>. Acesso em: 13 nov. 2015.

75 BRUN, R.; RADEMAKERS, F. ROOT – an object oriented data analysis framework. *Nucl. Instrum. Meth. A*, v. 389, n. 1–2, p. 81–86, 1997. doi: 10.1016/S0168-9002(97)00048-X.

76 MATIAS, P. *Novo método para assinatura e identificação de sinais de eletrocomunicação de peixes elétricos de campo fraco da espécie Gymnotus carapo*. 2011. p. 53. Dissertação (Mestrado em Física) – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2011. Disponível em: <http://www.teses.usp.br/teses/disponiveis/76/76132/tde-03052011-164310>. Acesso em: 17 nov. 2015.

77 ANALOG Devices. *8-channel, 1 MSPS, 8-/10-/12-bit ADCs with sequencer in 20-lead TSSOP*: AD7908/AD7918/AD7928. Norwood, MA, US, 2014. Disponível em: <http://www.analog.com/media/en/technical-documentation/data-sheets/AD7908_7918_7928.pdf>. Acesso em: 26 nov. 2015.

78 CENA, G.; CEREIA, M.; BERTOLOTTI, I. C.; SCANZIO, S.; VALENZANO, A.; ZUNINO, C. RTnet – a flexible hard real-time networking framework. In: IEEE CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 10., 2005, Catania, IT. *Proceedings...* Pitscataway, NJ, US: IEEE Press, c2005. p. 449–456. doi: 10.1109/ETFA.2005.1612559.

79 BUIT, E. *Real-time network performance characterization*: Locbus v.s. RTnet. Enschede, NL: Department of Electrical Engineering, University of Twente, 2004. Technical Report 56341. Disponível em: <http://doc.utwente.nl/56341>. Acesso em: 26 nov. 2015.

80 ANGUITA, D.; BONI, A.; RIDELLA, S. A digital architecture for support vector machines: theory, algorithm, and FPGA implementation. *IEEE T. Neural Networ.*, v. 14, n. 5, p. 993–1009, 2003. doi: 10.1109/TNN.2003.816033.

81 IRICK, K. M.; DEBOLE, M.; NARAYANAN, V.; GAYASEN, A. A hardware efficient support vector machine architecture for FPGA. In: INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES, 16., 2008, Palo Alto, CA, US. *Proceedings...* Los Alamitos, CA, US: IEEE Computer Society Press, c2008. p. 304–305. doi: 10.1109/FCCM.2008.40.

82 CADAMBI, S.; DURDANOVIC, I.; JAKKULA, V.; SANKARADASS, M.; COSATTO, E.; CHAKRADHAR, S.; GRAF, H. P. A massively parallel FPGA-based coprocessor for support vector machines. In: IEEE SYMPOSIUM ON FIELD PROGRAMMABLE CUSTOM COMPUTING MACHINES, 17., 2009, Napa, CA, US. *Proceedings...* Los Alamitos, CA, US: IEEE Computer Society Press, c2009. p. 115–122. doi: 10.1109/FCCM.2009.34.

83 ALTERA Corp. *Stratix IV FPGA vs. Virtex-5 logic efficiency*. San Jose, CA, US, 2012. Disponível em: <https://www.altera.com/products/fpga/features/stxiv-logic-efficiency.tablet.html>. Acesso em: 14 nov. 2015.

84 PAPADONIKOLAKIS, M.; BOUGANIS, C. A novel FPGA-based SVM classifier. In: INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE TECHNOLOGY, 2010,

Beijing, CN. *Proceedings...* Piscataway, NJ, US: IEEE Press, c2010. p. 283–286. doi: 10.1109/FPT.2010.5681485.

85 ORDAZ-MORENO, A.; DE JESUS ROMERO-TRONCOSO, R.; VITE-FRIAS, J. A.; RIVERA-GILLEN, J. R.; GARCIA-PEREZ, A. Automatic online diagnosis algorithm for broken-bar detection on induction motors based on discrete wavelet transform for FPGA implementation. *IEEE T. Ind. Electron.*, v. 55, n. 5, p. 2193–2202, 2008. doi: 10.1109/TIE.2008.918613.

86 BENKRID, A.; BENKRID, K.; CROOKES, D. Design and implementation of a generic 2D orthogonal discrete wavelet transform on FPGA. In: ANNUAL IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES, 11., 2003, Napa, CA, US. *Proceedings...* Los Alamitos, CA, US: IEEE Computer Society Press, c2003. p. 162–172. doi: 10.1109/FPGA.2003.1227252.

87 KUZUME, K.; NIIJIMA, K.; TAKANO, S. FPGA-based lifting wavelet processor for real-time signal detection. *Signal Process.*, v. 84, n. 10, p. 1931–1940, 2004. doi: 10.1016/j.sigpro.2004.06.020.

88 JAMIN, A.; MÄHÖNEN, P. FPGA implementation of the wavelet packet transform for high speed communications. In: INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, 12., 2002, Montpellier, FR. *Proceedings...* Berlin, DE: Springer, c2002. p. 212–221 (Lecture Notes in Computer Science, v. 2438). doi: 10.1007/3-540-46117-5_23.

89 HUANG, S.-J.; YANG, T.-M.; HUANG, J.-T. FPGA realization of wavelet transform for detection of electric power system disturbances. *IEEE T. Power Deliver.*, v. 17, n. 2, p. 388–394, 2002. doi: 10.1109/61.997905.

90 UZUN, I. S.; AMIRA, A. Real-time 2-D wavelet transform implementation for HDTV compression. *Real-Time Imaging*, v. 11, n. 2, p. 151–165, 2005. doi: 10.1016/j.rti.2005.01.001.

91 RITTER, J. *Wavelet based image compression using FPGAs*. 2002. p. 99. Dr. rer. nat. Dissertation – Martin-Luther-Universität Halle-Wittenberg, Halle, DE, 2002. Disponível em: <`http://sundoc.bibliothek.uni-halle.de/diss-online/02/03H033/prom.pdf`>. Acesso em: 13 nov. 2015.

92 CAPURRO, A.; MALTA, C. P. Noise autocorrelation and jamming avoidance performance in pulse type electric fish. *B. Math. Biol.*, v. 66, n. 4, p. 885–905, 2004. doi: 10.1016/j.bulm.2004.02.002.

93 BAFFA, O.; CORREA, S. L. Magnetic and electric characteristics of the electric fish *Gymnotus carapo*. *Biophys. J.*, v. 63, n. 2, p. 591–593, 1992. doi: 10.1016/S0006-3495(92)81633-8.