

Universidade de Brasília – UnB  
Faculdade UnB Gama – FGA  
Engenharia Eletrônica

**Estudo Comparativo da Implementação  
Coprocessada em Sistemas em Chip do  
Algoritmo de Treinamento do Classificador LDA  
Aplicado em Interfaces Cérebro-Máquina**

Autores: Heleno da Silva Moraes e Oziel da Silva Santos

Orientador: Dr. Marcus Vinícius Chaffim Costa

Coorientador: PhD. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2018





Heleno da Silva Moraes e Oziel da Silva Santos

**Estudo Comparativo da Implementação Coprocessada em  
Sistemas em Chip do Algoritmo de Treinamento do  
Classificador LDA Aplicado em Interfaces  
Cérebro-Máquina**

Monografia submetida ao curso de graduação  
em Engenharia Eletrônica da Universidade  
de Brasília, como requisito parcial para ob-  
tenção do Título de Bacharel em Engenharia  
Eletrônica.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Dr. Marcus Vinícius Chaffim Costa

Coorientador: PhD. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2018

---

Heleno da Silva Morais e Oziel da Silva Santos

Estudo Comparativo da Implementação Coprocessada em Sistemas em Chip do Algoritmo de Treinamento do Classificador LDA Aplicado em Interfaces Cérebro-Máquina/ Heleno da Silva Morais e Oziel da Silva Santos. – Brasília, DF, 2018-  
76 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Marcus Vinícius Chaffim Costa

Coorientador: PhD. Daniel Mauricio Muñoz Arboleda

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB  
Faculdade UnB Gama – FGA , 2018.

1. Sistemas em Chip. 2. LDA. I. Dr. Marcus Vinícius Chaffim Costa. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estudo Comparativo da Implementação Coprocessada em Sistemas em Chip do Algoritmo de Treinamento do Classificador LDA Aplicado em Interfaces Cérebro-Máquina

CDU 02:141:005.6

---

# Errata



Helena da Silva Moraes e Oziel da Silva Santos

# **Estudo Comparativo da Implementação Coprocessada em Sistemas em Chip do Algoritmo de Treinamento do Classificador LDA Aplicado em Interfaces Cérebro-Máquina**

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, DF, 06 de Julho de 2018 – Data da aprovação do trabalho:

---

**Dr. Marcus Vinícius Chaffim Costa**  
Orientador

---

**PhD. Daniel Mauricio Muñoz Arboleda**  
Coorientador

---

**Dr. Cristiano Jacques Miosso Rodrigues Mendes**  
Convidado 1

---

**MSc. Renato Coral Sampaio**  
Convidado 2

Brasília, DF  
2018





**Heleno da Silva Moraes**

*Ao meu pai, Joaquim Albino de Moraes †*

Oziel da Silva Santos

*Aos meus pais, José Antônio e Gezonita Maurício.*

# Agradecimentos

**Heleno da Silva Moraes**

—

Oziel da Silva Santos

# Resumo

As *Brain Computer Interfaces* (BCI) são sistemas capazes de realizar uma comunicação entre o cérebro e dispositivos eletrônicos. Cada vez mais estudadas por cientistas da área as BCIs já apresentam um grande número de aplicações. Um dos principais procedimentos para implementação de uma BCI é a classificação dos sinais gerados pelo cérebro. E a partir da classificação que os processos de comandos são executados. Existem inúmeros algoritmos que realizam este tipo de classificação, um deles é o classificador *Linear Discriminant Analysis* (LDA). Em 2010 o cientista francês Fabien Lotte publicou um trabalho onde realiza a implementação deste classificador, na plataforma *Matlab*, obtendo como melhor resultado de acurácia 96,43% na classificação de sinais de imagética motora, fornecidos pela competição *BCI Competition III*. Um dos pontos importantes e de maior necessidade de processamento para implementação deste classificador é processo de treinamento, onde são obtidos os hiperplanos capazes de separar as classes dos sinais em estudo. Estes hiperplanos são obtidos através de cálculos matriciais. Um dos sistemas capazes acelerar algoritmos que realizam este tipo de cálculo são os SoCs que contenham FPGA, onde são explorados o paralelismo de processos. Sendo assim, neste trabalho é apresentado um estudo da implementação em cálculos de ponto flutuante do algoritmo de treinamento do classificador LDA em um sistema coprocessado hardware-software utilizando-se do SoC *Zynq-7000* (onde se encontram embarcado um processador *ARM Cortex A9* dual core e um FPGA *Artix-7*), realizando a comparação com implementações em *Matlab* desenvolvida por Fabien Lotte e a implementação em um sistema embarcado utilizando Linguagem de programação C. Os resultados mostraram que o algoritmo implementado em linguagem C apresentou melhor desempenho computacional sendo ele 17 vezes mais rápido que o algoritmo executado em *Matlab*. Já o sistema coprocessado apresenta um melhor desempenho em funções de cálculo devido ao seu paralelismo, entretanto a latência do barramento de comunicação do sistema em hardware com o sistema em software é um limitante do seu desempenho. Sendo assim a melhor implementação para este algoritmo é a linguagem C.

**Palavras-chaves:** BCI; LDA; FPGA; SoC; Sistemas Embarcados.



# Abstract

**Key-words:** BCI; LDA; FPGA; SoC; Embedded Systems.





# Lista de ilustrações

Figura 1 – Três principais áreas do sistema nervoso central. Fonte: (KANDEL, 2013)	29
Figura 2 – Estrutura de um neurônio. Fonte: (SIULY, 2012)	30
Figura 3 – Sistema Internacional de Posicionamento 10-20. Fonte: (CAMPISI; ROCCA; SCARANO, 2012).	32
Figura 4 – Exemplos de diferentes tipos de EEG. Fonte: (CAMPISI; ROCCA; SCARANO, 2012)	33
Figura 5 – Diferença entre os sistemas ECoG e EEG. Fonte: (WOLPAW, 2012).	34
Figura 6 – Fluxograma de processos de uma BCI, que pode ser aplicado em mecanismos motorizados, reabilitação muscular, alarmes, entre outros. Adaptado de (WOLPAW, 2012).	34
Figura 7 – Diagrama simplificado da relação entre hardware e software em um SoC. Adaptado de (CAO et al., 2017)	39
Figura 8 – Placa de Desenvolvimento Zybo-Board. Fonte: (DIGILENT, 2017)	40
Figura 9 – Diagrama funcional do Zynq-7000 AP SoC	41
Figura 10 – Atividades realizadas pelo algoritmo de (LOTTE; GUAN, 2011).	43
Figura 11 – Atividades realizadas pelo algoritmo de treinamento do classificador LDA, desenvolvido por (LOTTE; GUAN, 2011).	44
Figura 12 – Representação do módulo somador/subtrator	46
Figura 13 – Bloco para cálculo de média com vinte somas em paralelo.	47
Figura 14 – Bloco para cálculo da matriz de covariância com quatro cálculos em paralelo.	48
Figura 15 – Fluxograma de implementação de hardware em um SoC. Adaptado de (CROCKETT et al., 2014)	49
Figura 16 – Fluxo de dados para leitura e escrita no barramento AXI Fone: (XILINX, 2011)	51
Figura 17 – Resultados obtidos com CSP-LDA	54
Figura 18 – Roteamento do circuito implementado referente a todo o sistema de coprocessamento.	57
Figura 19 – Legenda para a Figura 18.	57
Figura 20 – Estimativa do consumo energético do sistema.	58
Figura 21 – Simulação do IP de soma de 20 operandos.	59
Figura 22 – Simulação do IP de covariância para um único módulo.	60
Figura 23 – Sistema geral com IP	71



# Lista de tabelas

Tabela 1 – Número de tarefas classificadas e não classificadas por sujeito (BLANKERTZ et al., 2006). . . . .	36
Tabela 2 – Resultados obtidos por outros autores em implementações de algoritmos de classificação em FPGA. . . . .	42
Tabela 3 – Acurácia de classificação do algoritmo LDA utilizando do algoritmo CSP para maximização de variância de classes. (LOTTE; GUAN, 2011)	42
Tabela 4 – Acurácia da classificação usando o extrator CSP, classificador LDA e <i>Data-Set IVa - BCI Competition III</i> . . . . .	53
Tabela 5 – Acurácia da classificação dos sinais com treinamento em linguagem C.	54
Tabela 6 – Análise de perfil da função de treinamento. . . . .	55
Tabela 7 – Consumo de hardware da função Média. . . . .	55
Tabela 8 – Consumo de hardware da função Covariância. . . . .	56
Tabela 9 – Consumo de hardware do coprocessamento após implementação. . . . .	56
Tabela 10 – Acurácia da classificação dos sinais com treinamento em sistema coprocessado. . . . .	61
Tabela 11 – Propriedades das plataformas utilizadas. . . . .	61
Tabela 12 – Acurácia das implementações de estudo. . . . .	61
Tabela 13 – Tempos de execução da função de treinamento nas diferentes implementações. . . . .	62



# Lista de abreviaturas e siglas

BCI	<i>Brain Computer Interface</i> (Interface Cérebro Máquina)
EEG	Eletroencefalografia
SVM	<i>Support Vector Machine</i> (Máquina de Vetor de Suporte)
LDA	<i>Linear Discriminant Analysis</i> (Análise Discriminante Linear)
SoC	<i>System on Chip</i> (Sistema em Chip)
FPGA	<i>Field Programmable Gate Array</i> (Matriz de Portas Programáveis por Campo)
SNC	Sistema Nervoso Central
VHDL	<i>VHISC Hardware Description Language</i> (VHISC Linguagem de Descrição de Hardware)
VHISC	<i>Very High Speed Integrated Circuit</i> (Circuito Integrado de Alta Velocidade)
LUT	<i>Look Up Table</i> (Tabela da Verdade)
FF	Flip-Flop
DSP	<i>Digital Signal Processor</i> (Processador de Sinal Digital)
RAM	<i>Random Access Memory</i> (Memória de Acesso Aleatório)
MUX	Multiplexador
CSP	<i>Common Spatial Pattern</i> (Padrão Espacial Comum)
ECoG	Eletrocorticografia
SO	Sistema Operacional
RNA	Rede Neural Artificial
ADC	<i>Analog Digital Converter</i> (Conversor Digital Analógico)
ARM	<i>Advanced RISC Machine</i> (Máquina RISC Avançada)
AXI	<i>Advanced eXtensible Interface</i> (Interface Extensível Avançada)
FSM	<i>Finite State Machine</i> (Máquina de Estados Finitos)

IP	<i>Intellectual Properties</i> (Propriedade Intelectual)
SDK	<i>Software Development Kit</i> (Kit de Desenvolvimento de Software)
PS	<i>Processor Software</i> (Processador de Software)
PL	<i>Programmable Logic</i> (Lógica Programável)

# Lista de símbolos

$\mu$	Micro
$\theta$	<i>Theta</i>
$\alpha$	<i>Alpha</i>
$\beta$	<i>Beta</i>
$\gamma$	<i>Gamma</i>





# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>25</b>
<b>1.1</b>	<b>Justificativa</b>	<b>27</b>
<b>1.2</b>	<b>Objetivos</b>	<b>27</b>
1.2.1	Objetivos Gerais	27
1.2.2	Objetivos Específicos	27
<b>2</b>	<b>REFERENCIAL TEÓRICO</b>	<b>29</b>
<b>2.1</b>	<b>O Cérebro</b>	<b>29</b>
<b>2.2</b>	<b>Eletroencefalografia</b>	<b>31</b>
<b>2.3</b>	<b><i>Brain Computer Interface</i></b>	<b>33</b>
<b>2.4</b>	<b><i>BCI Competition</i></b>	<b>35</b>
2.4.1	<i>BCI Competition III</i>	36
2.4.2	<i>BCI Competition III - Dataset IVa</i>	36
<b>2.5</b>	<b><i>Linear Discriminant Analysis</i></b>	<b>37</b>
<b>2.6</b>	<b><i>System-on-Chip</i></b>	<b>38</b>
2.6.1	Aquitetura Simplificada de um SoC	39
2.6.2	<i>Zybo-Board</i>	39
2.6.2.1	Comunicação Entre PS e PL	40
<b>2.7</b>	<b>Estado da Arte</b>	<b>41</b>
<b>3</b>	<b>METODOLOGIA</b>	<b>43</b>
<b>3.1</b>	<b>Algoritmo Implementado</b>	<b>43</b>
<b>3.2</b>	<b>Implementação em Sistema Coprocessado</b>	<b>44</b>
3.2.1	Ferramentas Utilizadas	45
3.2.2	Metodologias de Desenvolvimento	45
3.2.2.1	Implementação em Hardware	45
3.2.3	Protocolo Para Análise estatística	49
<b>3.3</b>	<b>Implementação em software</b>	<b>50</b>
3.3.1	Métodos e Técnicas	50
<b>3.4</b>	<b>Integração</b>	<b>50</b>
<b>4</b>	<b>RESULTADOS E DICUSSÃO</b>	<b>53</b>
<b>4.1</b>	<b>Resultados obtidos em MATLAB</b>	<b>53</b>
<b>4.2</b>	<b>Resultados Obtidos em Linguagem C</b>	<b>54</b>
<b>4.3</b>	<b>Resultados Obtidos no Coprocessamento Hardware-Software</b>	<b>55</b>
4.3.1	Resultados Obtidos Pós Implementação em Hardware	55

4.3.1.1	IP Para Cálculo de Média . . . . .	55
4.3.1.2	IP Para Cálculo da Covariância . . . . .	56
4.3.1.3	Sistema Coprocessado . . . . .	56
4.3.2	Resultados de Simulação de Hardware . . . . .	58
4.3.2.1	IP Para Cálculo de Média . . . . .	58
4.3.2.2	IP Para Cálculo de Covariância . . . . .	59
4.3.3	Resultados de Execução . . . . .	60
4.3.4	Análise de Resultados de Desempenho de Processamento . . . . .	61
<b>5</b>	<b>CONCLUSAO . . . . .</b>	<b>63</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>65</b>
	<b>ANEXOS . . . . .</b>	<b>69</b>
	<b>ANEXO A – SISTEMA GERAL . . . . .</b>	<b>71</b>
	<b>ANEXO B – TUTORIAL PARA INSTALAÇÃO DO SISTEMA OPE- RACIONAL LINUX NO ZYBO . . . . .</b>	<b>73</b>
B.0.1	Geração do Arquivo u-boot.elf . . . . .	73
B.0.1.1	Requisitos Necessários . . . . .	73
B.0.1.2	Instruções . . . . .	73
B.0.2	Geração do Arquivo BOOT.bin . . . . .	74
B.0.2.1	Instruções . . . . .	74
<b>B.1</b>	<b>Compilando o Kernel Linux . . . . .</b>	<b>75</b>
B.1.1	Requisitos Necessários . . . . .	75
B.1.1.1	Instruções . . . . .	75

# 1 Introdução

Através de uma rede de mais de 100 bilhões de células nervosas interconectadas, o cérebro realiza o controle de nossas ações, percepções, emoções e etc (KANDEL, 2013). Estas células são chamadas de *neurônios*, e neles são armazenados sinais elétricos, que representam todas as informações de controle (SIULY; LI; ZHANG, 2017). Estes sinais podem ser medidos pela eletroencefalografia (EEG), que é um sistema de medição de sinais elétricos produzidos pelo cérebro durante atividades cerebrais (LOTTE; GUAN, 2011).

De acordo com (SIULY, 2012), a EEG é uma das mais importantes ferramentas para diagnosticar doenças cerebrais. Além do diagnóstico de doenças cerebrais uma outra aplicação para os sinais adquiridos pela EEG são as *Brain Computer Interfaces* (BCIs) (LOTTE; GUAN, 2011). Uma BCI é um sistema que realiza a comunicação entre o cérebro e um computador (SIULY; LI; ZHANG, 2017), onde sua principal função é a tradução dos sinais elétricos cerebrais em comandos de controle para qualquer dispositivo eletrônico (SIULY; LI; ZHANG, 2017).

A BCI realiza a tradução destes sinais através de seis passos: 1) medição dos sinais provenientes de atividades cerebrais, normalmente através da EEG, 2) pré-processamento destes sinais, 3) extração de características, 4) classificação, 5) tradução dos sinais em comandos e 6) realimentação (MASON; BIRCH, 2003). Um dos principais passos para a implementação de uma BCI é a *classificação*, pois é após este passo que é realizada a tradução dos sinais provenientes da EEG em comandos de controle (MASON; BIRCH, 2003).

Em aprendizado de máquina e reconhecimento de padrões, a classificação é caracterizada como um algoritmo que atribui parte de um sinal de entrada a um número de classes ou categorias (BRUNELLI, 2009). Um exemplo é a classificação de um e-mail como spam ou não-spam. Os algoritmos que realizam a classificação dos sinais de entrada são chamados de **classificadores** (SIULY; LI; ZHANG, 2017). De acordo com (LOTTE, 2008, p. 41), "estes classificadores são capazes de aprender como identificar um vetor de características, graças aos processos de treinamento".

O algoritmo que realiza a classificação é caracterizado por uma função matemática que mapeia um sinal de entrada em sua respectiva classe (LOTTE, 2008). Os classificadores preferidos pelos pesquisadores são os **classificadores supervisionados**. Estes classificadores fazem uso de um conjunto de dados para realizar o processo de treinamento do classificador. O conjunto de dados de treinamento é formado por vetores de características previamente atribuídos às suas respectivas classes (LOTTE, 2008). Portanto os

classificadores supervisionados são implementados a partir de dois processos: *treinamento* e *testes* (SIULY; LI; ZHANG, 2017). As *Support Vector Machines* (SVMs), em português, Máquinas de Vetor de Suporte, os *Linear Discriminant Analysis* (LDAs), em português, Análise Discriminante Linear, as Redes Neurais Artificiais (RNAs) e as árvores de decisões são alguns exemplos mais conhecidos de classificadores do tipo supervisionados (SIULY; LI; ZHANG, 2017).

O LDA como dito anteriormente é um dos classificadores supervisionados e tem como principais vantagens a simplicidade e atratividade computacional, por se tratar de um classificador linear (THEODORIDIS; KOUTROUMBAS et al., 1999). O objetivo do LDA é usar uma transformação linear para encontrar um conjunto otimizado de vetores discriminantes e remapear o conjunto de características original, em um outro conjunto de dimensão inferior (SHASHOA et al., 2016).

A simplicidade e robustez do LDA possibilita sua implementação em um sistema embarcado, o que viabiliza sua portabilidade considerando as restrições de desempenho computacional e consumo energético dos Sistemas em Chip (SoCs - do inglês *System on Chip*).

Um SoC é caracterizado pela implementação de todo um sistema computacional, composto por: memórias, processadores, entradas e saídas, conversores de dados, controladores de periféricos, entre outros, em um único chip de silício (CROCKETT et al., 2014). Diferente dos computadores tradicionais, que possuem seu sistema implementado a partir de módulos isolados e combinados em uma placa de circuito impresso, ou placa-mãe, os SoCs possuem como principais características um baixo custo de implementação, além de baixo consumo de potência, menor tamanho físico, maior confiabilidade e, dependendo dos recursos disponíveis, maior desempenho computacional, se comparado com um computador tradicional (CROCKETT et al., 2014). Um exemplo de um SoC é a plataforma *Zynq* que combina em um único chip processadores *Advanced Risc Machine* (ARM) e *Field Programmable Gate Arrays* (FPGA), conversores analógicos digitais (ADC - do inglês) (CROCKETT et al., 2014).

Tendo em vista a grande vantagem dos SoCs sobre os computadores tradicionais, onde são implementados e executados os algoritmos de classificação, este trabalho apresenta um estudo da viabilidade da implementação em hardware e em software embarcado, do algoritmo de treinamento do classificador LDA desenvolvido previamente por (LOTTE; GUAN, 2011) na plataforma *Matlab*, realizando a comparação de consumo computacional (hardware), desempenho computacional (tempo de execução) e consumo energético entre as implementações em hardware e software. Em particular a implementação em hardware consiste no mapeamento do algoritmo na FPGA da plataforma *Zynq*, afim de paralelizar seus processos, enquanto que a implementação em software consiste em executar o algoritmo em um sistema embarcado utilizando os cores ARM, também da plataforma *Zynq*,

além da comparação com sua implementação inicial em *Matlab*.

## 1.1 Justificativa

As aplicações das BCIs apresentam um crescente desenvolvimento graças ao aumento do interesse em pesquisas voltadas para o tema (BLANKERTZ et al., 2006). As BCIs utilizam-se de algoritmos de classificação para realizar o processo de tradução dos sinais cerebrais em comandos de controle (MASON; BIRCH, 2003). O LDA é um tipo de classificador utilizado para realizar tal processo. Um dos principais processos para implementação de um classificador é o processo de treinamento, é com este processo que o classificador é capaz de classificar corretamente os sinais de entrada, quando realizado um bom processo de treinamento (LOTTE; GUAN, 2011). Por se tratar de um algoritmo linear e conseqüentemente não exigir um grande esforço computacional, torna-se viável sua implementação em um sistema embarcado. Como os SoCs apresentam características de baixo consumo de energia e tamanho físico pequeno, a implementação de algoritmos de classificação em sistema deste porte podem tornar as BCIs mais acessíveis, tendo em vista que um algoritmo de treinamento embarcado em um SoC reduzirá a necessidade de um sistema computacional tradicional, além de um melhor processamento computacional, pois sua implementação em plataformas FPGAs possibilitam a paralelização de seus processos (CROCKETT et al., 2014).

## 1.2 Objetivos

### 1.2.1 Objetivos Gerais

O objetivo geral do presente trabalho é implementar o algoritmo de treinamento de um classificador LDA em um sistema embarcado coprocessado (hardware/software), utilizando um SoC na plataforma *Zynq* da *Xilinx*, no intuito de estudar os ganhos em relação ao tempo de execução se comparado com implementações tradicionais.

### 1.2.2 Objetivos Específicos

- Reproduzir os resultados obtidos por (LOTTE; GUAN, 2011) no desenvolvimento do algoritmo de treinamento do classificador LDA;
- Implementar em sistema embarcado o algoritmo de treinamento utilizando os cores ARM da *Zynq*;
- Realizar uma análise de *profile* para determinação de funções que exigem um maior esforço computacional;

- Mapear a(s) função(ões) que exigem um maior esforço computacional em hardware FPGA utilizando a linguagem VHDL;
- Validar as implementações utilizando as bases de dados do *BCI Competition III*, em específico o conjunto de dados BCI III *dataset IVa*.
- Realizar uma análise estatística do erro associado a ambas implementações, comparadas com a implementação na plataforma *Matlab*.
- Realizar uma análise de ganhos ou perdas de processamento computacional, comparando a implementação em software com o coprocessamento;

## 2 Referencial Teórico

Este capítulo apresenta os conceitos teóricos que abordam este trabalho, detalhando a estrutura cerebral responsável pela geração dos sinais de interesse na subseção 2.1, o sistema de captura dos sinais (a EEG) na subseção 2.2, o sistema que se refere às BCIs que realizam a tradução dos sinais em comandos para dispositivos na subseção 2.3, os detalhes técnicos a respeito dos sinais oferecidos pela base de dados *BCI Competition* na subseção 2.4, a estrutura geral de um classificador LDA na subseção 2.5, os conceitos de um SoC na subseção 2.6 e por fim o estado da arte das implementações de algoritmos de classificação na subseção 2.7.

### 2.1 O Cérebro

O Sistema Nervoso Central (SNC) é o responsável direto pelo comando do nosso comportamento geral ([CLARK NASHAAT BOUTROS, 2005](#)). Ele pode ser dividido em três principais áreas: tronco encefálico ou medula espinhal, o cérebro e o cerebelo (Figura 1) ([ALVAREZ; LEMOS, 2006](#)).

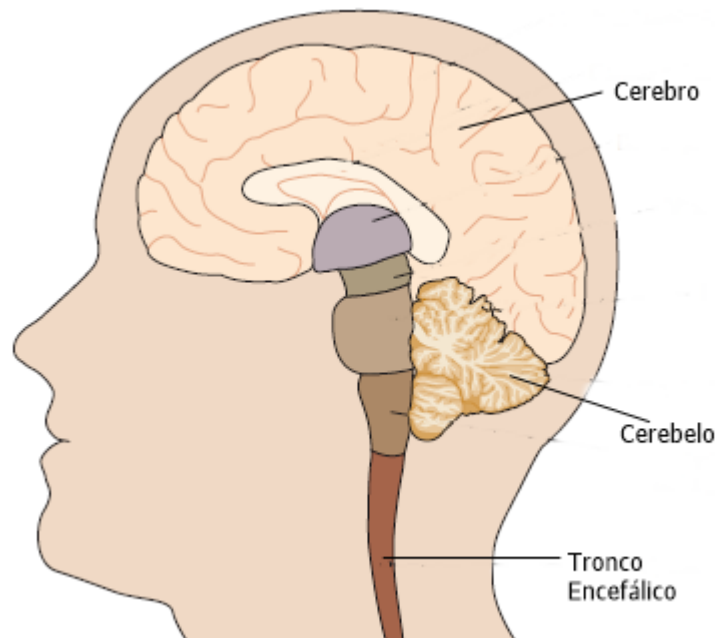


Figura 1 – Três principais áreas do sistema nervoso central. Fonte: ([KANDEL, 2013](#))

O tronco encefálico, parte caudal do SNC, recebe e processa todos os sinais dos sensores corporais, além de realizar o controle dos membros e do tronco humano ([KANDEL, 2013](#)).

O cérebro é o processador do SNC, nele são recebidos e processados os sinais da medula espinhal, além de fornecer todos os sinais de controle para a própria medula, que por sua vez, distribui os sinais para os membros e para o tronco (KANDEL, 2013).

O cerebelo está localizado logo atrás do tronco encefálico, onde, através de fibras chamadas de pedúnculos, realizam a comunicação entre si (KANDEL, 2013). O cerebelo é a segunda maior estrutura do SNC contendo mais da metade dos neurônios cerebrais (SIULY, 2012). É ele o responsável nossa percepção (ALVAREZ; LEMOS, 2006), além do controle da região motora e a memória de movimentos (SIULY, 2012; ALVAREZ; LEMOS, 2006)

Os sinais de controle e de sensoriamento são sinais elétricos armazenados nos neurônios (KANDEL, 2013). Suas características eletroquímicas permitem que os mesmos armazenem e transmitam sinais elétricos para qualquer outra célula receptora mesmo que a longa distâncias (SIULY, 2012). Sua estrutura pode ser dividida em três principais partes: 1) Corpo da célula, 2) axônio e 3) dendrito, conforme apresentado na Figura(2).

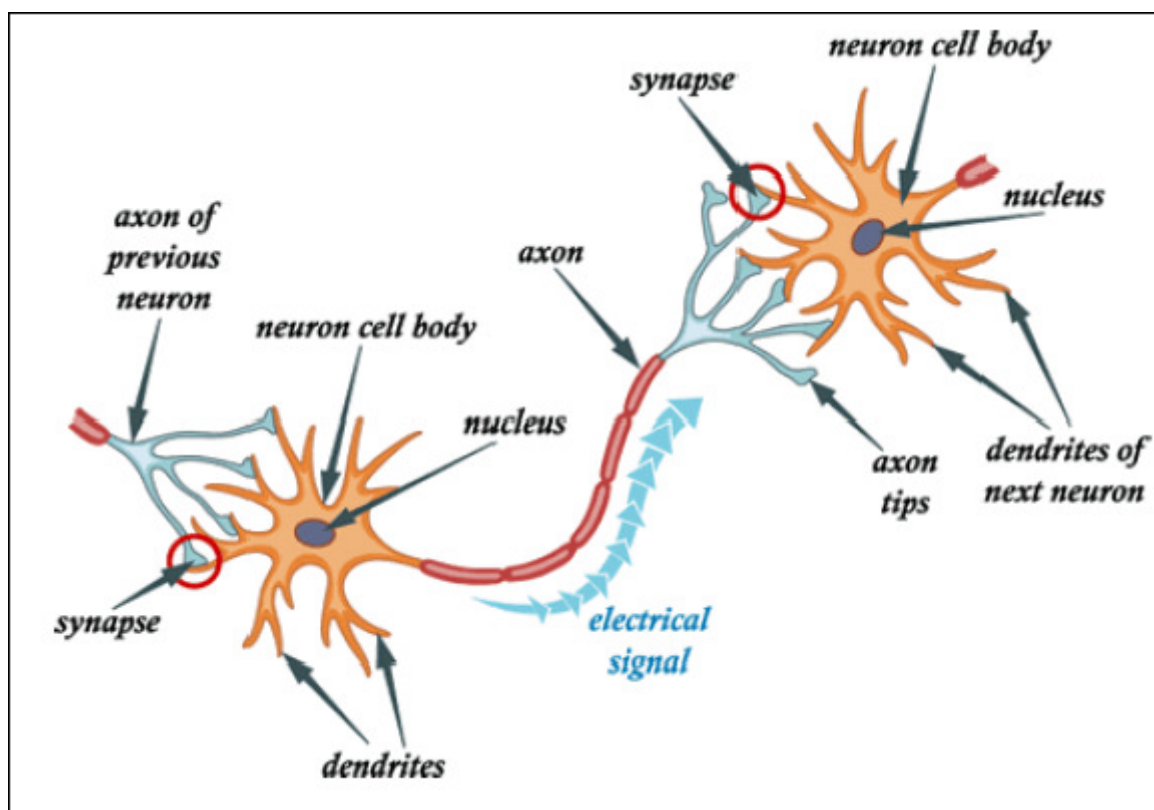


Figura 2 – Estrutura de um neurônio. Fonte: (SIULY, 2012)

A ativação dos neurônios é dada por meio de um gradiente de concentração eletroquímico, resultando na produção de uma corrente elétrica cerebral que flui através do axônio, o que torna possível a comunicação com outras células (SIULY, 2012). A corrente elétrica cerebral consiste comumente de íons de  $\text{Na}^+$ ,  $\text{K}^+$ ,  $\text{Ca}^{++}$  e  $\text{Cl}^-$  (TEPLAN et al., 2002).



As atividades elétricas cerebrais podem ser divididas em dois conjuntos: os potenciais de ação (AP) e os potenciais de sinapse (SP) (SIULY, 2012).

Sempre que um potencial de sinapse atinge o limite de condução, ou seja, se o potencial é carregado (estimulado por alguma atividade cerebral) até o ponto em que se é gerada uma corrente elétrica no axônio, um AP é iniciado (SIULY, 2012).

Como dito antes, o cerebelo é o responsável pelo controle da percepção e dos movimentos, além da memória de movimento, portanto a percepção, os movimentos e a imagética motora são os estímulos que iniciam uma AP na região do cerebelo (ALVAREZ; LEMOS, 2006).

Os potenciais elétricos gerados a partir dos potenciais de sinapse são armazenados na escalpe (ou couro cabeludo), onde durante uma atividade cerebral é criado um dipolo elétrico entre os dendritos e a soma (região ao redor do núcleo) (SIULY, 2012).

## 2.2 Eletroencefalografia

A Eletroencefalografia (EEG) é um sistema de aquisição de potenciais elétricos, que refletem atividades cerebrais humanas (SIULY; LI; ZHANG, 2017). É muito usado por profissionais de saúde e cientistas para avaliar e estudar funções cerebrais e diagnosticar distúrbios neurológicos (SIULY; LI; ZHANG, 2017).

A técnica popular que registra sinais do cérebro usando eletrodos estrategicamente colocados sobre o couro cabeludo do paciente, portanto não invasiva, é muito importante nos diagnósticos de doenças neurológicas (RAO, 2013). Em um exame de EEG, dependendo da sua aplicação, podem ser usados de 1 a 256 eletrodos a fim de captar paralelamente os sinais (RAO, 2013). Cada par de eletrodos formam um canal, portanto com 256 eletrodos tem-se uma leitura de 128 canais, também conhecido como EEG multi canais (RAO, 2013). Cada canal recebe um amplificador de instrumentação e um equipamento de gravação do EEG (RAO, 2013).

Devido às diferentes camadas e tecidos interpostos entre a fonte do sinal (que é a atividade neural do córtex) e os sensores colocados no couro cabeludo, tudo isso atuando como um filtro passa-baixa, faz com o que a resolução espacial do EEG seja ruim, em contra partida tem boa resolução temporal na faixa de milisegundos (RAO, 2013).

Em um adulto normal, o sinal típico de EEG tem sua amplitude variando de 1 a 100 microvolts, se for medido usando eletrodos de tipo agulha o seu módulo pode variar de 10 a 20 milivolts. Considerando a não uniformidade do cérebro humano e a organização funcional do córtex, o sinal de EEG pode variar bastante de acordo com a disposição dos eletrodos (SIULY, 2012).

Com a baixa amplitude desse tipo de sinal, ele pode sofrer facilmente atenuações,

contaminações em seu espectro, como por exemplo interferência da rede elétrica de 60 Hz e seus harmônicos associados, e principalmente atividades musculares exercidas no ato da extração dos sinais. Portanto no momento do exame os pacientes são orientados a não realizar nenhum tipo de movimento (RAO, 2013).

O sistema internacional 10-20 é utilizado para especificar um padrão de alocação dos eletrodos na cabeça dos pacientes ou voluntários. Esse padrão determina as distâncias entre os eletrodos (SIULY; LI; ZHANG, 2017). São posicionados eletrodos primeiramente em pontos estratégicos, os pontos são nomeados por: *mastoids*, eletrodos de referência localizados atrás de cada orelha (A1 e A2), *Nasion*, referência no segmento do topo do nariz, porém nivelado com os olhos e por fim, o *onion*, referência situada na base do crânio no ponto médio da parte de trás da cabeça, as medidas são feitas a partir desses pontos, no sentido transversal e médio, com intervalos de 10 e 20 % (RAO, 2013), assim representado na Figura 3.

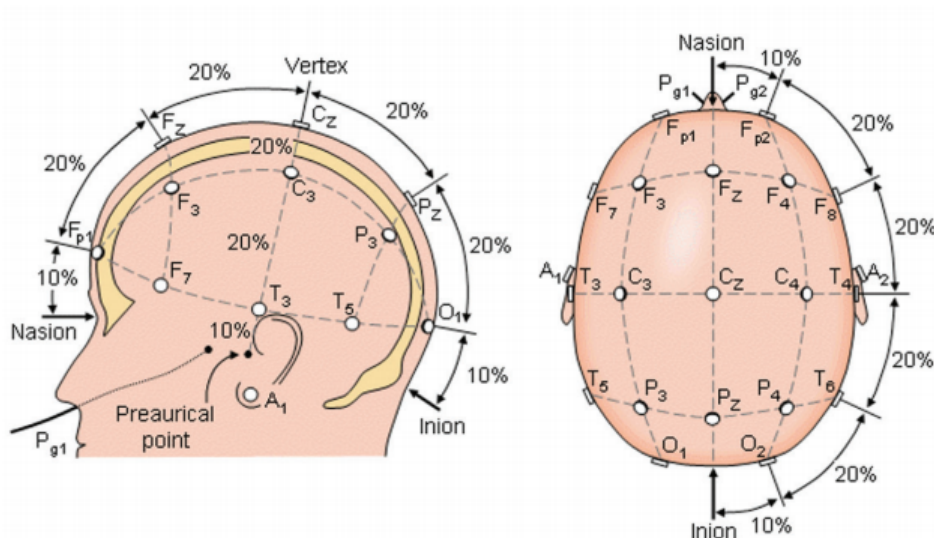


Figura 3 – Sistema Internacional de Posicionamento 10-20. Fonte: (CAMPISI; ROCCA; SCARANO, 2012).

A Frequência do sinal é um dos parâmetros mais importantes na avaliação clínica de anomalias em EEGs, e também para compreender o comportamento operacional na pesquisa cognitiva (SIULY, 2012). Com inúmeras oscilações não periódicas, milhares de comunicação inter-neurônios e comportamentos estocásticos o EEG humano é comumente ordenado em faixas de frequências específicas. Tais bandas são divididas em: delta (0.5-4 Hz) , theta( $\theta$ )(4-8Hz), alpha( $\alpha$ )(8-13Hz), beta( $\beta$ )(13-30Hz) e acima de 30Hz gama( $\gamma$ ), conforme Figura 4.

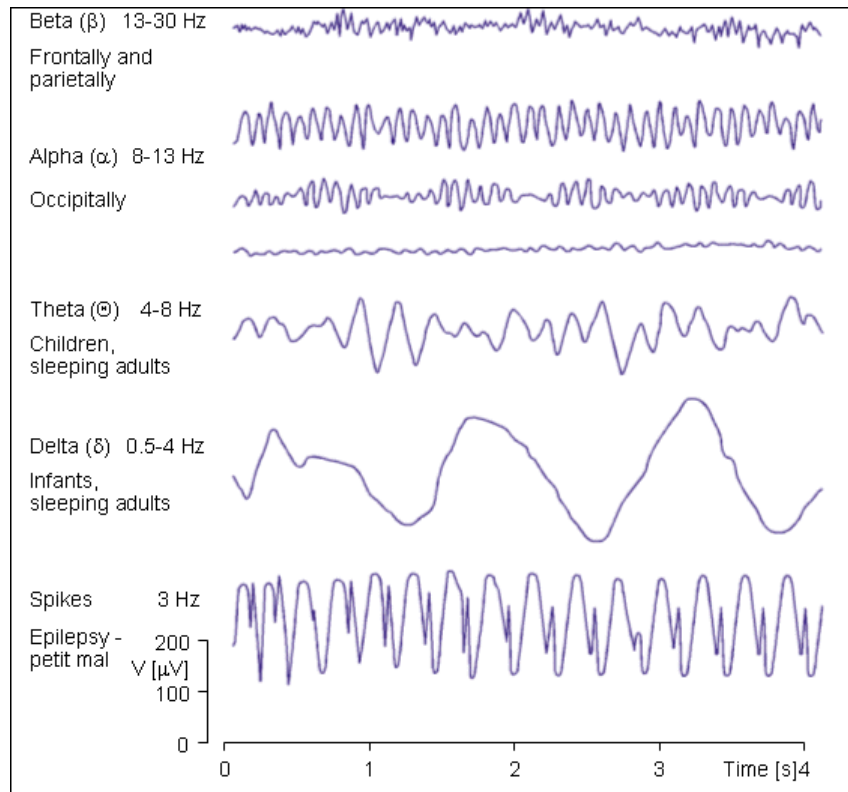


Figura 4 – Exemplos de diferentes tipos de EEG. Fonte: (CAMPISI; ROCCA; SCARANO, 2012)

## 2.3 Brain Computer Interface

Uma das aplicações dos sinais adquiridos pela EEG são as *Brain Computer Interface* (em português, Interface cérebro máquina) (LOTTE; GUAN, 2011; SIULY, 2012). Uma Interface Cérebro Máquina (BCI, do inglês *Brain Computer Interface*) é um sistema que analisa sinais cerebrais e os convertem para um novo sinal de saída (WOLPAW, 2012), ou seja, uma BCI é um sistema que realiza a comunicação entre o cérebro e o mundo externo sem a interação neuromuscular (WOLPAW, 2012).

O termo BCI foi utilizado pela primeira vez por Jacques Vidal em 1970, onde foram utilizadas técnicas invasivas para aquisição dos sinais cerebrais, sistema de eletrocorticografia (ECoG) (WOLPAW, 2012), sistema em que os sensores de aquisição eram instalados na região do córtex. Em 1980 o mesmo Jacques Vidal publicou pela primeira vez a utilização de uma BCI não invasiva utilizando o sistema da EEG (GUGER et al., 2013), as duas técnicas são apresentadas na Figura 5.

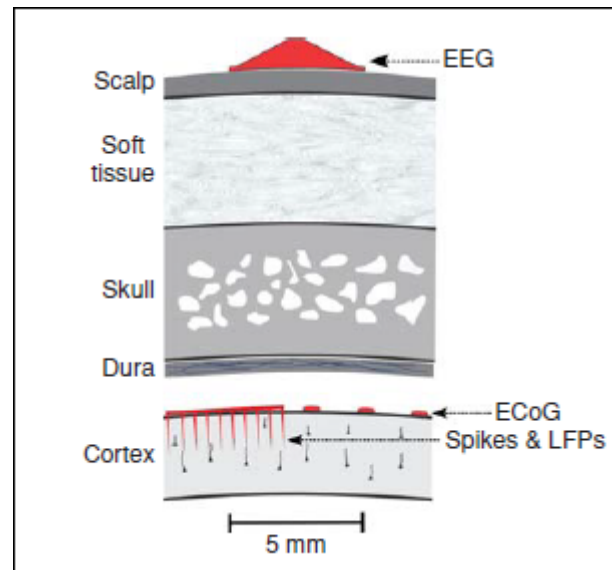


Figura 5 – Diferença entre os sistemas ECoG e EEG. Fonte: (WOLPAW, 2012).

Atualmente é mais comum a utilização do sistema da EEG para aquisição de sinais cerebrais por se tratar de um sistema não invasivo (GUGER et al., 2013). A BCI realiza a tradução dos sinais cerebrais por meio de seis passos, de acordo com o fluxograma apresentado na Figura 6.

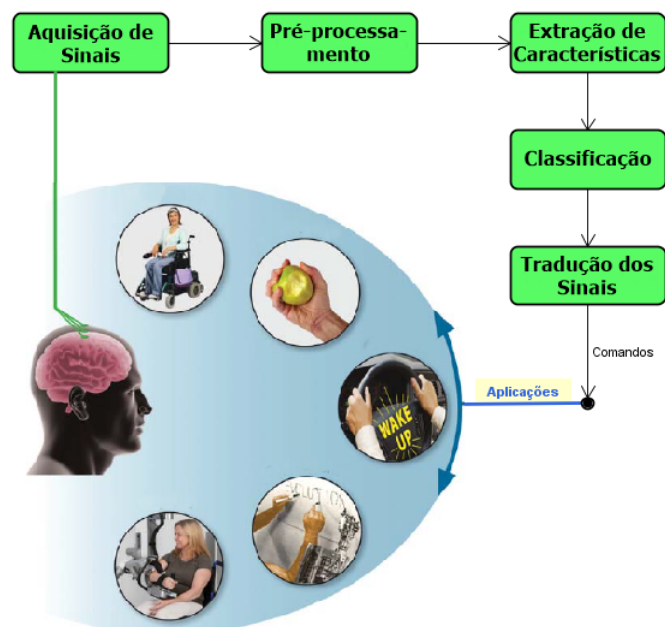


Figura 6 – Fluxograma de processos de uma BCI, que pode ser aplicado em mecanismos motorizados, reabilitação muscular, alarmes, entre outros. Adaptado de (WOLPAW, 2012).

1. **Aquisição de sinais:** Consiste na primeira etapa de um BCI, na qual são aferidos potenciais elétricos provenientes de atividades cerebrais (SIULY, 2012). Os sinais

utilizados neste trabalho foram adquiridos utilizando a EEG e provenientes de uma base de dados de acesso livre, discutida adiante, na seção 2.4.

2. **Pré-Processamento:** Nesta etapa são utilizados filtros para atenuar a presença de ruídos no sinal e amplificar as informações relevantes dentro do sinal (SIULY, 2012).
3. **Extração de Características:** Nesta etapa são extraídos dos sinais alguns parâmetros relevantes, chamados de características (SIULY, 2012).
4. **Classificação:** Nesta etapa as características extraídas no processo anterior são rotuladas em determinadas classes (SIULY, 2012).
5. **Tradução em comandos:** Nesta etapa um comando é associado a cada uma das respectivas classes (SIULY, 2012).
6. **Realimentação:** Por fim é fornecido ao usuário da BCI uma informação a respeito do seu estado mental, qual atividade cerebral foi detectada (SIULY, 2012).

Um dos principais passos para a implementação de uma BCI é a **classificação**, pois é após este passo que é realizada a tradução dos sinais provenientes da EEG em comandos de controle (MASON; BIRCH, 2003). Para isso são utilizados algoritmos classificadores, dentre os quais um tipo de algoritmo de classificação é o LDA, a ser abordado na seção 2.5.

São utilizadas estratégias mentais para definir ao usuário tarefas mentais afim de gerar características padronizadas nos sinais cerebrais de acordo com determinada tarefa, para que o classificador possa interpretá-las corretamente (SIULY, 2012). Uma das abordagens mais comuns é imagética motora, que caracteriza a imaginação do movimento de um membro do corpo humano (SIULY, 2012).

## 2.4 BCI Competition

A *BCI Competition* é uma competição que promove o desenvolvimento e melhoria da tecnologia voltada para as BCIs, sendo submetidas diferentes técnicas de análise de dados cerebrais (BLANKERTZ et al., 2006). Já foram realizadas quatro edições da competição, nos anos de 2001, 2002, 2004 e 2008 (BLANKERTZ et al., 2006). Em cada uma destas competições são fornecidos publicamente sinais cerebrais, adquiridos em laboratórios especializados (BLANKERTZ et al., 2006). Estes sinais são divididos em dois conjuntos de dados, os dados de treinamento e os dados de teste, que são utilizados para treinamento e teste dos algoritmos dos participantes (BLANKERTZ et al., 2006).

### 2.4.1 BCI Competition III

O objetivo do *BCI Competition III* é validar as metodologias de classificação e processamento de sinais cerebrais aplicados em BCIs desenvolvidas pelos participantes da competição (BLANKERTZ et al., 2005). Esta edição foi realizada entre Maio e Junho de 2004, período no qual foram disponibilizados 8 *datasets* (I, II, IIIa, IIIb, IVa, IVb, IVc e V), desenvolvidos com a participação de 49 laboratórios especializados (BLANKERTZ et al., 2006). Para cada um dos *datasets* foram realizadas diferentes tarefas que estimulam atividades cerebrais durante a aquisição dos sinais, configurando assim um objetivo específico para cada um dos *datasets* (BLANKERTZ et al., 2005).

### 2.4.2 BCI Competition III - Dataset IVa

O *dataset IVa* refere-se a um conjunto de dados adquiridos por meio da EEG, em que os sujeitos (indivíduos nos quais foram capturados os sinais) foram submetidos a estimular o cérebro por imagética motora, através de indicações visuais (BLANKERTZ et al., 2006). Os indivíduos foram submetidos a realizarem três tarefas, indicadas visualmente por 3.5s cada tarefa, sendo interrompidas em períodos aleatórios entre 1.75s e 2.25s, onde o sujeito era submetido a um período de relaxamento (BLANKERTZ et al., 2006). As três tarefas de imagéticas motoras foram: (L) mão esquerda, (R) mão direita e (F) pé direito (BLANKERTZ et al., 2006).

Os sinais foram adquiridos de 5 sujeitos rotulados como *aa*, *al*, *av*, *aw* e *ay*, onde foram executadas no total 280 tarefas por cada sujeito. Algumas destas tarefas foram previamente classificadas, ou seja, foram mapeadas características do sinal cerebral em sua respectiva classe, formando o conjunto de dados de treinamento, enquanto que as tarefas não classificadas formam o conjunto de dados de teste (BLANKERTZ et al., 2005). Estes sinais foram adquiridos, pré-processados e disponibilizados por *Fraunhofer FIRST, Intelligent Data Analysis Group (Head: Klaus-Robert Müller), and Charité University Medicine Berlin, Campus Benjamin Franklin, Department of Neurology, Neurophysics Group* (BLANKERTZ et al., 2006). A Tabela 1 apresenta a quantidade de tarefas previamente classificadas (nomeados #tr) e a quantidade de tarefas não classificadas (nomeadas #te) para cada sujeito.

Tabela 1 – Número de tarefas classificadas e não classificadas por sujeito (BLANKERTZ et al., 2006).

Sujeitos	#tr	#te
<i>aa</i>	168	112
<i>al</i>	224	56
<i>av</i>	84	196
<i>aw</i>	56	224
<i>ay</i>	28	252

Os dados *Data Set IVa* foram adquiridos e armazenados utilizando amplificadores do tipo *BrainAmp* e uma capa de eletrodos de 128 canais. Foram utilizados 118 canais de EEG posicionados de acordo com o sistema 10-20. Cada um destes canais foram filtrados em banda passante, utilizando um filtro *butterworth* de quinta ordem entre as frequências de 0,05 e 200 Hz, posteriormente foram digitalizados com uma frequência de amostragem de 1 kHz com precisão de 16 bits, apresentando uma resolução de 0,1  $\mu\text{V}$ , além disso também foram disponibilizados os mesmos dados com uma frequência de amostragem de 100 Hz (BLANKERTZ et al., 2005).

## 2.5 Linear Discriminant Analysis

O LDA é um classificador desenvolvido para explorar as informações no reconhecimento de padrões supervisionados, as informações conhecidas são contidas num vetor de treinamento previamente disponibilizado (IZENMAN, 2008). No algoritmo do LDA as informações de maior relevância são descobertas, enquanto que as de menor são eliminadas. O critério usado pelo algoritmo é obter as dimensões que possuem as características mais distintas das classes padrão (KORKMAZ et al., 2017).

Para um melhor entendimento, supõe-se a existência de um conjunto de dados  $\vec{x}$ , em que  $\vec{x} = (x_1, x_2, \dots, x_n)^T$  e  $T$  indica a transposição, com características multivariadas, e que cada dado seja conhecido devido ser proveniente de uma das classes  $y$ , tal que, são predefinidas com características semelhantes aos dados. As classes podem ser exemplificadas como sendo: espécies de plantas, presença ou ausência de uma condição médica específica, diferentes tipos de tumores, tipos de veículos automotores entre outros. Para separar as classes conhecidas uma das outras, é atribuído um rótulo a cada classe, então os dados são representados como dados rotulados (IZENMAN, 2008).

Devido a indispensabilidade de diminuir as dimensões dos dados de um determinado conjunto, o objetivo do LDA é reduzir a dimensão do espaço de conjunto de dados, resolvendo o inconveniente da sobreposição (SINGH; PRAKASH; CHANDRASEKARAN, 2016).

O LDA tem a proposta de encontrar uma transformação ótima para maximizar a proporção de acordo com a Equação 2.1. com isso encontrando o vetor  $W$  que proporciona a melhor discriminação (KETSUWAN; PADUNGWEANG, 2017).

$$J(W) = \frac{W^T S_B W}{W^T S_W W} \quad (2.1)$$

$S_B$  é a matriz de dispersão entre as classes e  $S_W$  a matriz de dispersão dentro das classes.



$S_B$  é caracterizado pela matriz de covariância entre as classes, obtido segundo a Equação 2.2

$$S_B = \frac{\sum_{j=1}^c (m_j - \bar{m})(m_j - \bar{m})^T}{n - 1} \quad (2.2)$$

$c$  é o número de classes, e  $\bar{m}$  o vetor média e  $m_j$  é o vetor dos dados pertinentes à classe  $j$  e  $x$  é o vetor de treinamento.

$$\bar{m} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.3)$$

$$m_j = \frac{1}{n_j} \sum_{x \in c_j} x \quad (2.4)$$

A matriz de dispersão entre classes é definida pela equação 2.5,

$$S_B = \sum_{j=1}^c \sum_{x \in C_j} (X - m_j)(X - m_j)^T \quad (2.5)$$

Para  $S_W$  sendo uma matriz não-singular, a solução da equação 2.1, pode ser escrita como:

$$S_W^{-1} S_B W = \lambda W \quad (2.6)$$

onde  $\lambda$  é o auto-valor correspondente ao auto-vetor  $W$ .

Ou seja, a proposta do classificador LDA é encontrar o hiperplano de maior separação entre as classes analisadas. O cálculo deste hiperplano é feito a partir dos dados de treinamento. Com os dados de treinamento, a distribuição de probabilidade é conhecida, podendo ser representada pela média da dispersão dos sinais. Em outras palavras os hiperplanos são obtidos através da correlação inter-classes e intra-classe, maximizando a separabilidade das classes e minimizando a variabilidade dentro da classe.

## 2.6 System-on-Chip

O termo Sistema em Chip (SoC, do inglês *System-on-Chip*), implica que todo sistema que contém funcionalidades implementadas em hardware e software se encontra em um único chip de silício, combinando processamento, lógica de alta velocidade, interface, memória entre outros componentes ao invés de uma implementação maior em vários chips físicos diferentes agrupados em uma placa-mãe (CROCKETT et al., 2014).

São vários os argumentos a favor da escolha de um SoC a uma arquitetura que utiliza placa-mãe, pode-se citar que a solução é de menor custo, viabiliza transferências de dados mais rápidas e seguras entre vários elementos do sistema, possui maior velocidade geral do sistema, menor consumo de energia entre vários outros elementos que fortalecem a escolha de um SoC em sistemas discretos com componentes equivalentes (CROCKETT et al., 2014).



### 2.6.1 Arquitetura Simplificada de um SoC

O conjunto da arquitetura pode se dividir em dois sistemas, sistema de hardware e sistema de software.

No **Sistema de Hardware** encontram-se todos os periféricos, memórias e processadores, para conecta-los existe um barramento de comunicação responsável por isso. Já no **Sistema de Software**, o software funciona sobre o processador, que também sustenta os aplicativos (geralmente em um Sistema Operacional) e também possui uma camada num nível mais baixo de software que faz a interface com o sistema de hardware. Um diagrama de blocos simplificado de um SoC é apresentado na Figura 7 (CROCKETT et al., 2014).

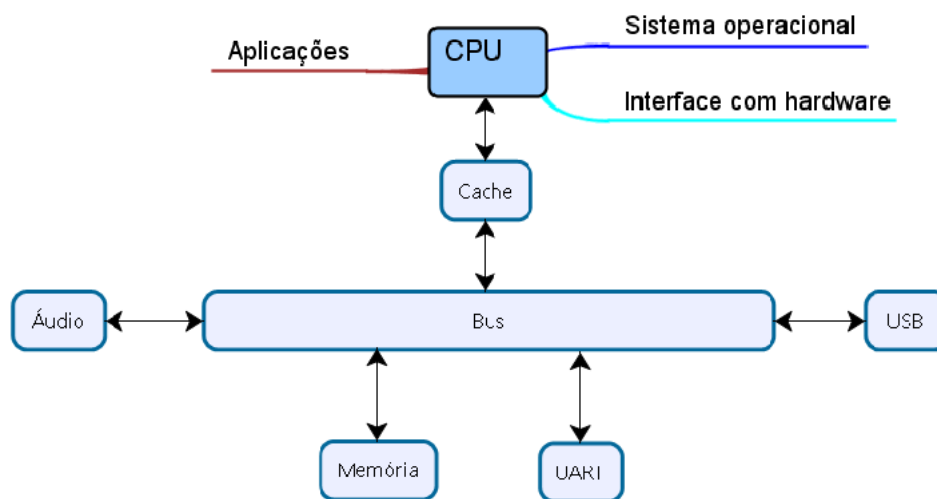


Figura 7 – Diagrama simplificado da relação entre hardware e software em um SoC. Adaptado de (CAO et al., 2017)

### 2.6.2 Zybo-Board

A Zybo é uma plataforma de desenvolvimento (Figura 8). Baseada na arquitetura Xilinx<sup>®</sup> All-Programmable System-on-Chip (AP SoC) do tipo Z-7010, o qual possui em seu encapsulamento um processador ARM Cortex-A9 de dois núcleos e um FPGA da família Xilinx 7-Series. Esse chip combinado com memórias, entradas e saídas de áudio e vídeo, USB, Ethernet, controlador de memória SD entre outros periféricos, proporciona um *kit* para desenvolvedores que procuram uma plataforma de desenvolvimento SoC com flexibilidade e boa capacidade de processamento (DIGILENT, 2017).

A Figura 9 mostra o diagrama funcional do Zynq-7000 AP SoC, o *Processing System* e a *Programmable Logic* têm sistemas de alimentação individual, permitindo o usuário habilitá-los ou não para gerenciamento de energia.

O Zynq-7000 AP SoC é composto pelos seguintes blocos majoritários

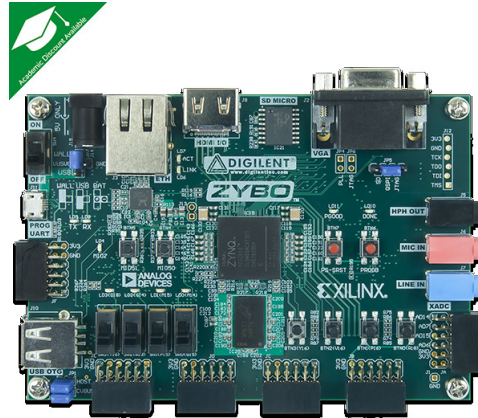


Figura 8 – Placa de Desenvolvimento Zybo-Board. Fonte: (DIGILENT, 2017)

- Sistema de Processamento (PS do inglês *Processing System*)
  - Unidade de Processador de Aplicação (APU - do inglês *Application Processor Unit*)
  - Interfaces de Memória (do inglês *Memory interfaces*)
  - Periféricos de entrada (IOP do inglês *I/O peripherals*)
  - Interconexões (do inglês *Interconnect*)
- Lógica Programável (PL - do inglês *Programmable Logic*)
  - *Look-Up Tables* - LUT São tabelas da verdade capazes de implementarem quaisquer funções lógicas combinacionais.
  - Flip-Flop Unidade de armazenamento que quando combinado com a LUT é utilizado para implementar circuitos sequenciais.
  - BRAM Bloco de memória de acesso randômico utilizado para armazenar dados.
  - DSP Processador digital de sinais, basicamente é uma unidade lógica aritmética.

#### 2.6.2.1 Comunicação Entre PS e PL

Conforme apresentado na figura 9 a PL realiza a comunicação com o PS através da interface de comunicação AXI-32bits. O AXI é uma das partes da família de protocolos *ARM-AMBA*. Em 2010 a ultima versão desenvolvida do AMBA (versão 4.0), foi incluída a segunda versão do AXI, o AXI4 (XILINX, 2011).

Existem três tipos de barramento AXI, o AXI4, AXI4-Lite e AXI4-Stream.

- AXI4 - usado quando os requisitos de projeto necessitam de alta performance no mapeamento de memória.

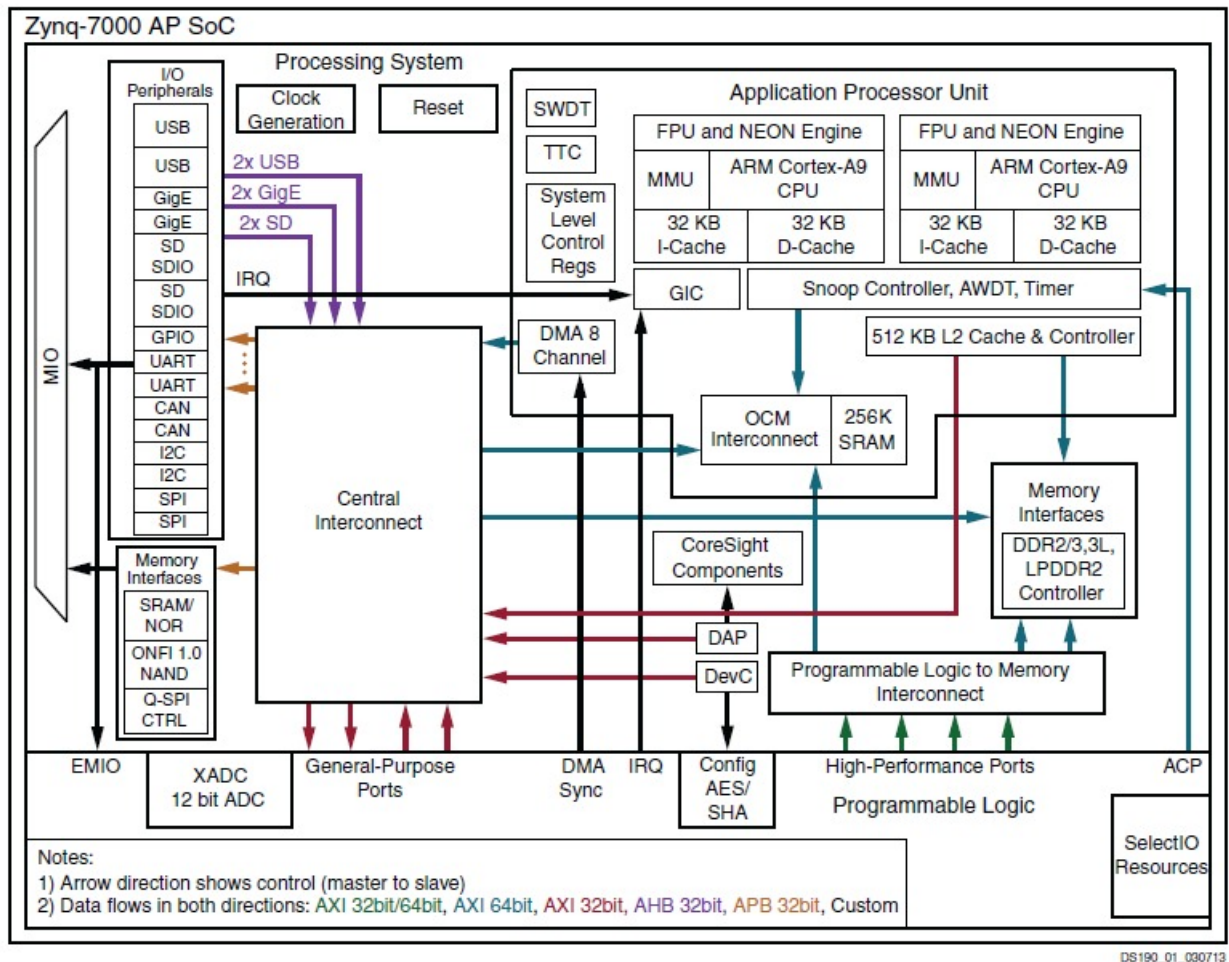


Figura 9 – Diagrama funcional do Zynq-7000 AP SoC

- AXI4-Lite usado para um simples mapeamento de memória.
- AXI4-Stream usado quando necessita-se fluxo de dados em alta velocidade.

## 2.7 Estado da Arte

A implementação de algoritmos de classificação em FPGA já foi relatada anteriormente na literatura, sendo utilizados para diferentes aplicações. A Tabela 2 apresenta a relação de alguns autores que realizaram a implementação de classificadores em SoCs utilizando FPGA, apresentando a acurácia obtida por cada autor, bem como os sinais de entrada e sua respectiva aplicação.

Os melhores resultados obtidos com a base de dados *BCI Competition - Dataset IV* foram de (WANG et al., 2004), onde utilizando do classificador LDA, conseguiu acurácias de 94.17% geral, 95.5% sobre o sujeito (aa), 100.0% sobre o sujeito (al), 80.6% sobre o sujeito (av) 100.0% sobre o sujeito (aw) 97.6% sobre o sujeito (ay).

Tabela 2 – Resultados obtidos por outros autores em implementações de algoritmos de classificação em FPGA.

<b>Autores</b>	<b>Algoritmo</b>	<b>Acurácia</b>	<b>Sinal</b>
( <i>YANG, 2003</i> )	RNA	92%	Imagem
( <i>IRICK et al., 2008</i> )	SVM	88.6%	Imagem
( <i>GLETTE; TORRESEN, 2009</i> )	EHW	<b>97.5%</b>	Imagem
( <i>GLETTE; TORRESEN, 2007</i> )	EHW	91.4 %	Ondas Sonoras
( <i>ALKIM; KILIÇ, 2011</i> )	LVQ	85%	EMG
( <i>YUAN et al., 2017</i> )	SVM	<b>97%</b>	Ondas Ultrassônicas
( <i>CHEN et al., 2016</i> )	NB	92%	EMG
( <i>KAIS et al., 2014</i> )	LDA	72%	EEG

A Tabela 3 apresenta a acurácia obtida por (*LOTTE; GUAN, 2011*) no ano de 2010, no desenvolvimento do algoritmo de classificação LDA, utilizando do algoritmo *Common Spatial Pattern* (CSP) para maximizar a variância do filtro passa-faixa utilizados pelo EEG para uma das classes, enquanto minimiza a variância para as outras demais classes, utilizando a base de dados *BCI Competition - Dataset IVa*.

Tabela 3 – Acurácia de classificação do algoritmo LDA utilizando do algoritmo CSP para maximização de variância de classes. (*LOTTE; GUAN, 2011*)

<b>Sujeitos</b>	<b>Acurácia do LDA com algoritmo CSP</b>
<i>aa</i>	66,70%
<i>al</i>	96,43%
<i>av</i>	47,45%
<i>aw</i>	71,88%
<i>ay</i>	49,60%

## 3 Metodologia

Neste capítulo é apresentado como foi realizada a implementação do algoritmo de treinamento do classificador LDA em um sistema coprocessado hardware-software, detalhando todos os módulos que compõem a implementação.

### 3.1 Algoritmo Implementado

Em 2010 o francês Fabien Lotte publicou um trabalho com objetivo de comparar as implementações de algoritmos de extração de características de sinais provenientes de atividades cerebrais, além de propor um novo algoritmo para extração de características dos sinais (LOTTE; GUAN, 2011).

Para coleta de resultados de seu trabalho, Lotte utilizou o algoritmo de classificação LDA e a base de dados *BCI Competition III - Dataset IVa*, obtendo como melhor resultado o extrator de características CSP, conforme apresentado anteriormente na Tabela 3.

O algoritmo foi desenvolvido utilizando a plataforma *Matlab*, onde foram utilizados os recursos e funções da plataforma. A Figura 10 apresenta um diagrama de atividades que descreve as funções utilizadas no algoritmo.

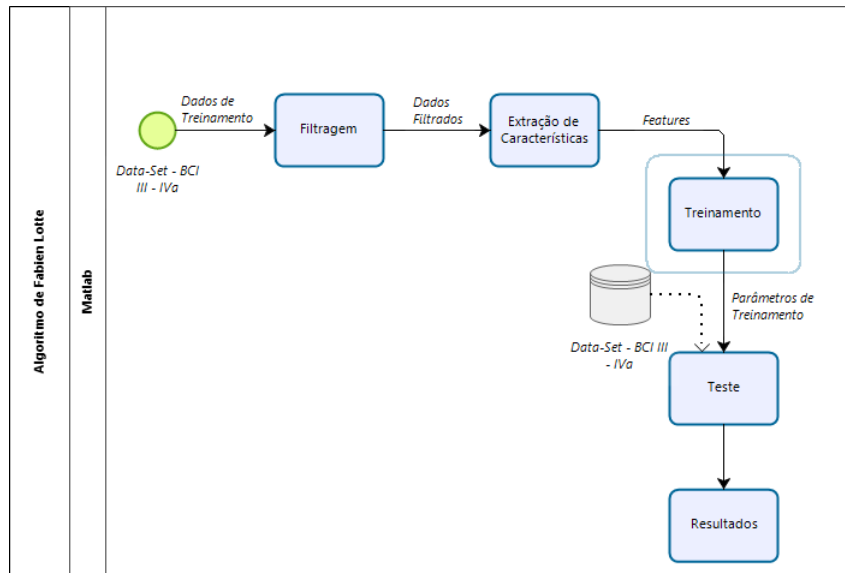


Figura 10 – Atividades realizadas pelo algoritmo de (LOTTE; GUAN, 2011).

Utilizando-se dos dados de treinamento do *BCI Competition III - Data Set -IVa*, o algoritmo inicia-se realizando a filtragem dos sinais a partir de um filtro *Butterworth* passa faixas de 5ª ordem, mantendo os sinais das faixas  $\alpha$  e  $\beta$  e atenuando as demais frequências. Em seguida, é realizado o processo de extração de características utilizando o algoritmo

CSP. Logo após, é realizado o processo de treinamento, onde são calculados hiperplanos que melhor separam as classes. Estes hiperplanos são os parâmetros utilizados no processo de classificação, sendo então, um dos parâmetros de entrada da função de teste, onde são classificados os sinais de testes, também fornecidos pelo *BCI Competition III - Data Set -IVa*. Por fim, são disponibilizados os resultados de acurácia e tempo de treinamento.

### 3.2 Implementação em Sistema Coprocessado

Conforme apresentado, o objetivo deste trabalho é o estudo dos ganhos ou perdas no processamento do algoritmo de treinamento do classificador LDA implementado em um coprocessamento hardware-software, utilizando-se do SoC *Zynq* embarcado no kit de desenvolvimento *Xilinx Zybo Board* modelo 70-10, comparado com a implementação realizada por (LOTTE; GUAN, 2011) e a implementação em software executado sobre um SO Linux embarcado nos processadores ARM, também disponíveis no SoC. Sendo assim, o trabalho se restringiu à implementação das atividades que compõem a função de treinamento do classificador LDA (atividades destacadas) na Figura 10, em um sistema coprocessado.

A Figura 11 apresenta um diagrama de atividades que descreve a função de treinamento do classificador LDA.

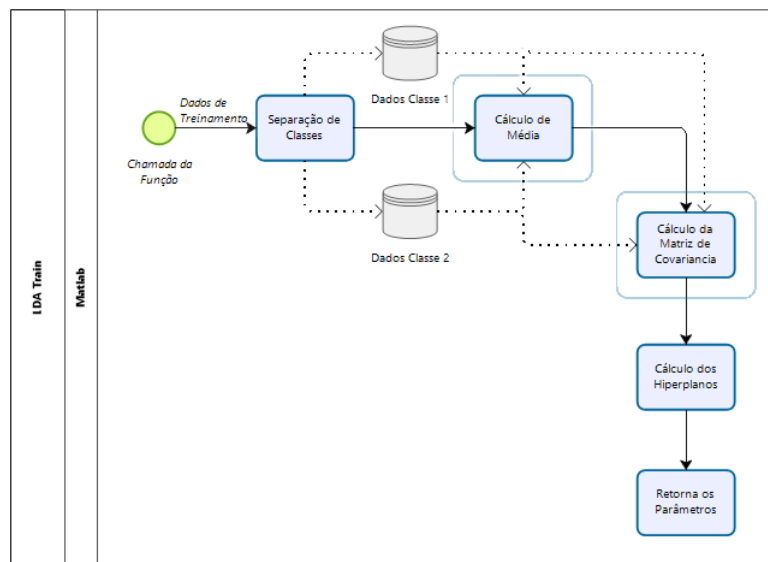


Figura 11 – Atividades realizadas pelo algoritmo de treinamento do classificador LDA, desenvolvido por (LOTTE; GUAN, 2011).

Inicialmente é realizada a separação dos sinais em dois conjuntos de dados que representam os dados referentes às duas classes em estudo. Em seguida são calculadas as médias dos sinais de cada classe. Estes valores são repassados para a função de cálculo da matriz de covariância (ou dispersão), onde são realizados cálculos matriciais conforme apresentado na Equação 2.2. Posteriormente são calculados os hiperplanos a partir da

covariância das características dos sinais. Estes hiperplanos representam os parâmetros de treinamento do classificador. São a partir deles que os sinais de teste são classificados.

Na realização do coprocessamento as funções de cálculo de média e cálculo de covariância são as duas funções que apresentam um maior esforço computacional. Portanto ambas as funções foram mapeadas em hardware, utilizando-se da linguagem VHDL, a fim de acelerar o algoritmo de treinamento explorando o paralelismo dos algoritmos, enquanto as demais funções foram compiladas em software na linguagem C, executada através de um SO Linux embarcado nos processadores do SoC utilizado neste projeto.

### 3.2.1 Ferramentas Utilizadas

Para estudo das implementações foram utilizadas cinco ferramentas principais:

- Software *Matlab R2016a - Student License* - Utilizado para reproduzir os resultados obtidos por (LOTTE; GUAN, 2011);
- SO Linux - Utilizado como ambiente de desenvolvimento de software embarcado;
- Software *Vivado - v.2017.4* - Utilizado para desenvolver, integrar e sintetizar os IP's de cálculo em ponto flutuante das funções média e covariância;
- *Software Development Kit (SDK) - v.2017.4* - Utilizado para desenvolver o arquivo fsbl.elf e para compilar o arquivo BOOT.bin.

Para implementação em software e implementação coprocessada utilizamos o SoC *Zynq* embarcado no kit de desenvolvimento *Zybo Board*.

### 3.2.2 Metodologias de Desenvolvimento

#### 3.2.2.1 Implementação em Hardware

Para implementação das funções de cálculo de média e cálculo da matriz de covariância em FPGA foi adotada a metodologia *bottom-up*, na qual cada sub-bloco desenvolvido foi testado antes de ser inserido ao bloco principal, também conhecido como *Top module*. Os cálculos foram realizados através dos blocos de propriedade intelectual (IP) desenvolvidos por (MUÑOZ et al., 2010). Os IPs realizam cálculos matemáticos em unidades de ponto flutuante de acordo com o padrão IEEE-754 (ELECTRICAL; ENGINEERS, 2008) com registradores de 27 bits, representados com:

- Expoente: 8 bits;

- Mantissa: 18 bits;
- Sinal: 1 bit.

O IP que realiza a operação de soma ou subtração é controlado por uma máquina de estado finita (FSM - do inglês *Finite State Machine*) (MUNOZ et al., 2010). O bloco apresentado na Figura 12 representa o IP somador/subtrator, com suas respectivas entradas e saídas.

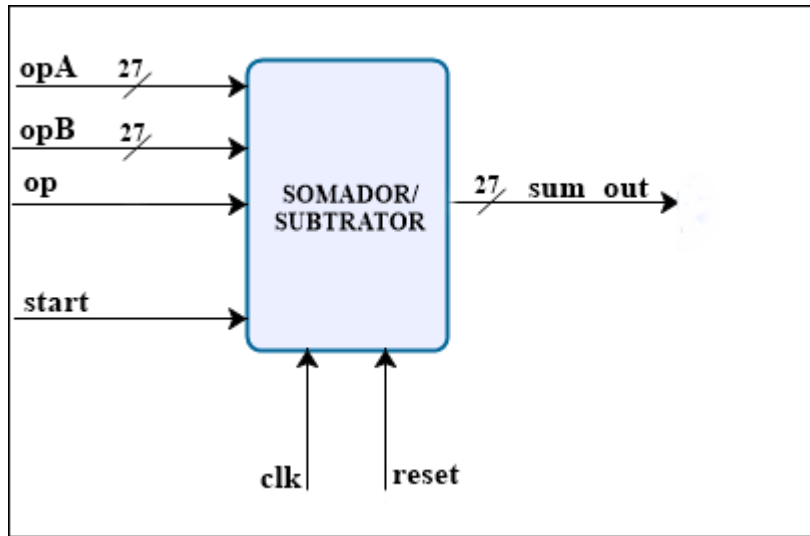


Figura 12 – Representação do módulo somador/subtrator

O sinal de entrada *op* representa se a função do bloco será soma (caso valor lógico for igual a 0) ou subtração (caso valor lógico for igual a 1) (MUNOZ et al., 2010). Os sinais *opA* e *opB* representam os valores de entrada, no caso da adição representam os valores da soma, e no caso da subtração representam minuendo e subtraendo, respectivamente. Há também a entrada *start* utilizado para iniciar a operação assim que seu nível lógico realiza uma transição de borda de subida (de nível lógico 0 para nível lógico 1), além das entradas de clock e reset.

Foram criados componentes para paralelizar o processo dos cálculos, buscando o maior nível de paralelismo. A partir do bloco de IP da Figura 12, para a função média foram implementados um total de 20 (vinte) somas em paralelo. Para sequenciar a soma final dos 20 operandos de entrada foi implementado um *dataflow* de somadores, onde a saída apresenta o resultado da soma final de 20 operandos (Figura 13). Já para a função de covariância foram implementados 4 componentes em paralelo que calculam parte da Equação 2.2 conforme apresentado na Figura 14.

Foram instanciados 5 blocos de somadores de 4 entradas, nomeados de soma primária, desenvolvidos a partir do bloco do IP somador da Figura 12. A saída de 4 (quatro) destes 5 (cinco) blocos são entradas de um novo bloco somador de 4 entrada, nomeado



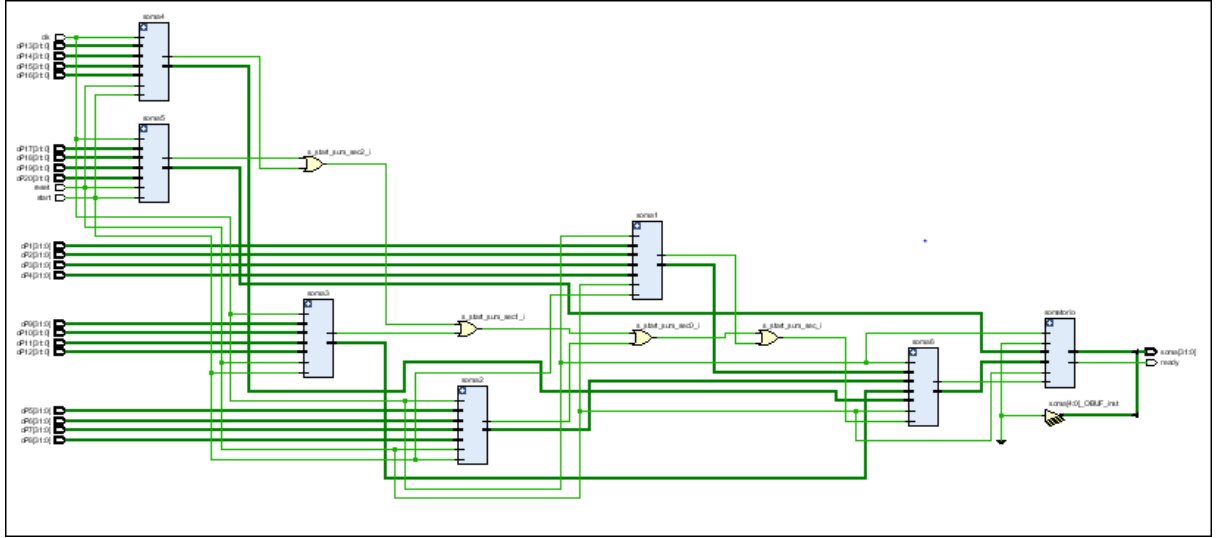


Figura 13 – Bloco para cálculo de média com vinte somas em paralelo.

de soma secundária, com *start* acionado após finalizar as somas primárias. A saída deste somador representa a entrada de um somador básico com 2 (duas) entradas, nomeado de soma final, completando a soma com a saída do 5<sup>o</sup> (quinto) somador primário, resultando na saída final a soma dos 20 (vinte) operandos de entrada. Após a soma de todos os valores pertencentes ao vetor de características de sua respectiva classe, o valor da soma total é dividido pela quantidade de sinais de características, resultando assim na média final de um determinado vetor de característica.

A matriz de covariância é encontrada a partir de um cálculo matricial, conforme anteriormente apresentado na Equação 2.2. O cálculo implementado em hardware realiza o cálculo interno ao somatório, sendo então a Equação 3.1 implementada em hardware:

$$cov_{hw} = \frac{(m_j - \bar{m})(m_j - \bar{m})}{n - 1} \quad (3.1)$$

A transposição da matriz (operator  $^T$ ) e o somatório foi mantido em software. Sendo assim, o módulo apresenta 5 (cinco) saídas paralelas para cada uma das entradas da equação 3.1.

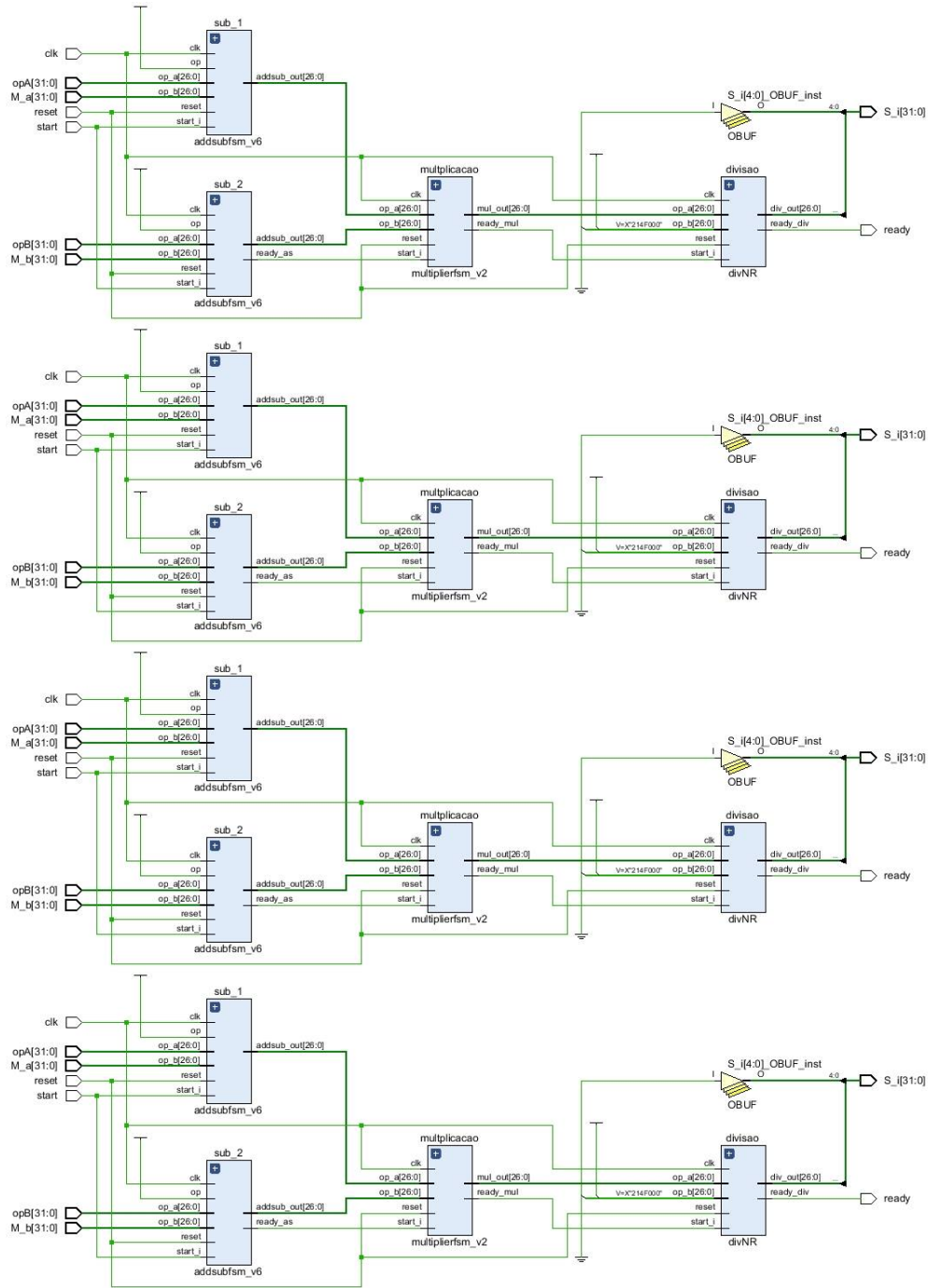


Figura 14 – Bloco para cálculo da matriz de covariância com quatro cálculos em paralelo.

A técnica utilizada na implementação do projeto foi a *dataflow*. O princípio básico das arquiteturas *dataflow* é que a execução das operações é acionada em função da disponibilidade dos operandos. Ela não segue uma estrutura convencional das máquinas algorítmicas, onde uma parte de controle é responsável pela sincronização das operações (SIQUEIRA, 2016). No caso dos sistemas em hardware implementados os blocos iniciam sua operação apenas quando suas entradas já estão carregadas.

Todos os módulos desenvolvidos foram atribuídos a um novo IP com barramento

AXI4-Lite para realizar a comunicação com o processador ARM. Onde são transmitido e recebido os dados de entrada referentes aos IPs de cálculo de média e cálculo da matrix de covariância. O diagrama geral da implementação, onde é apresentado a conexão entre ARM e todos os seus periféricos, além dos IPs desenvolvidos é apresentado no ANEXO A deste trabalho.

O fluxo de desenvolvimento do projeto em hardware utilizado neste trabalho é apresentado na Figura 15.

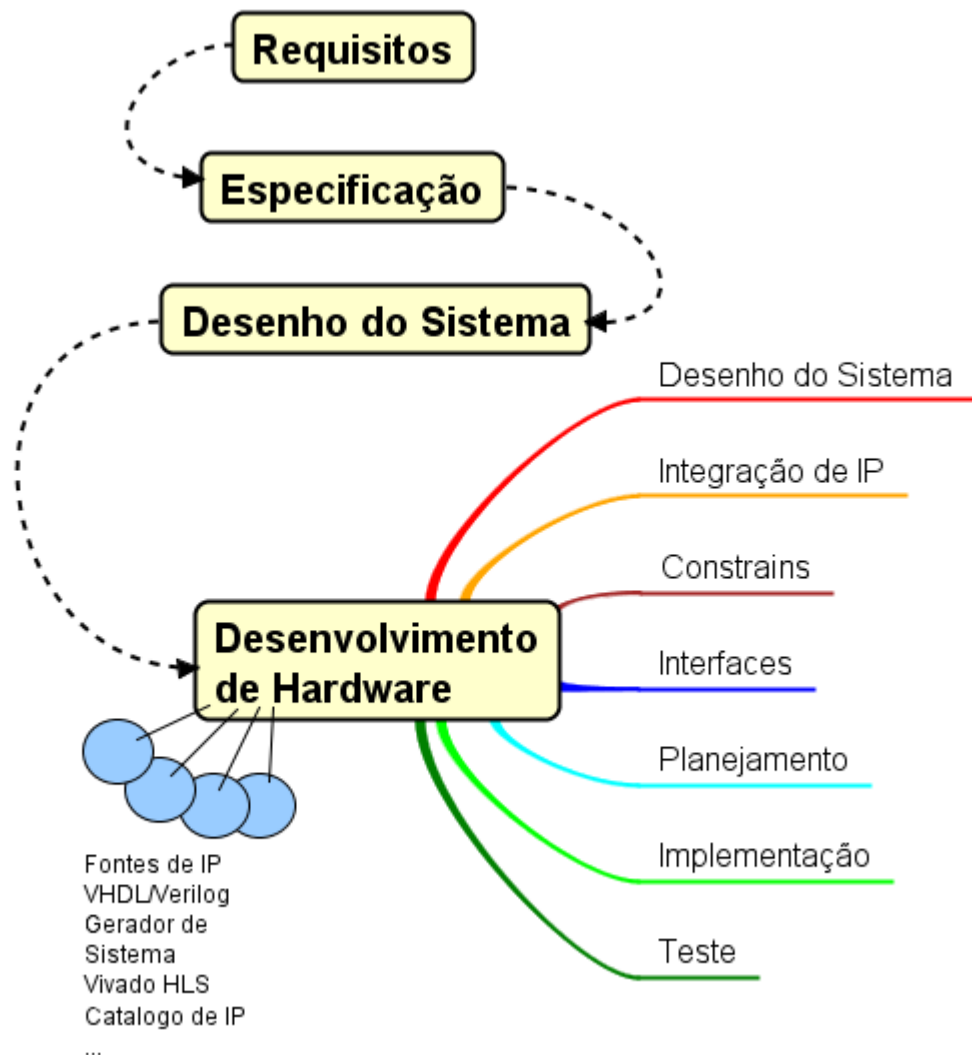


Figura 15 – Fluxograma de implementação de hardware em um SoC. Adaptado de (CROCKETT et al., 2014)

### 3.2.3 Protocolo Para Análise estatística

Para efeito de análise estatística comparatória foram coletados as seguintes informações pós implementação:

- Consumo de hardware: LUTs, FFs, blocos de DSP, blocos de memória RAM, I/O;

- Dados de desempenho: frequência de operação e tempo de execução;
- Dados simulados: Latência e *throughput*;
- Estimação do consumo de energético;
- Estimação estatística do erro quadrático de implementação em hardware comparada com a implementação original em *Matlab*.

### 3.3 Implementação em software

#### 3.3.1 Métodos e Técnicas

Para o auxílio desta implementação, foi instalado o sistema operacional *Debian Linux* sobre os cores ARM, cujos recursos e passos necessários para a instalação do SO estão contidos no ANEXO B.

O processo para implementação seguiu o método *bottom-up*, blocos de códigos menores foram implementados e testados separadamente, com a finalidade de todos os blocos serem integrados e testados em um único bloco principal. Todas as codificações foram implementadas utilizando a linguagem de programação C, e para compilar o código principal foi utilizado o compilador GCC (*GNU Compiler Collection*). Com esse método conseguimos realizar uma análise de perfil das funções implementadas onde foram reportados o tempo de execução de cada uma das funções, sendo assim possível deduzir as funções que necessitam de maior esforço computacional. As entradas para programa desenvolvido são os sinais de treinamento fornecidos pelo *dataset IVa* do *BCI Competition III*.

Como parâmetro de análise estatística foram coletados os seguintes dados:

- Consumo de memória
- Tempo de execução
- Erro da implementação em comparação aos resultados obtidos por (LOTTE; GUAN, 2011)

### 3.4 Integração

Toda a integração do projeto coprocessado é possibilitado pelo uso do barramento *AXI4-Lite*, que interliga a PL com a PS, um esquemático do processo de escrita e leitura nos IP através do barramento *AXI4-Lite* pode ser observado na Figura 16

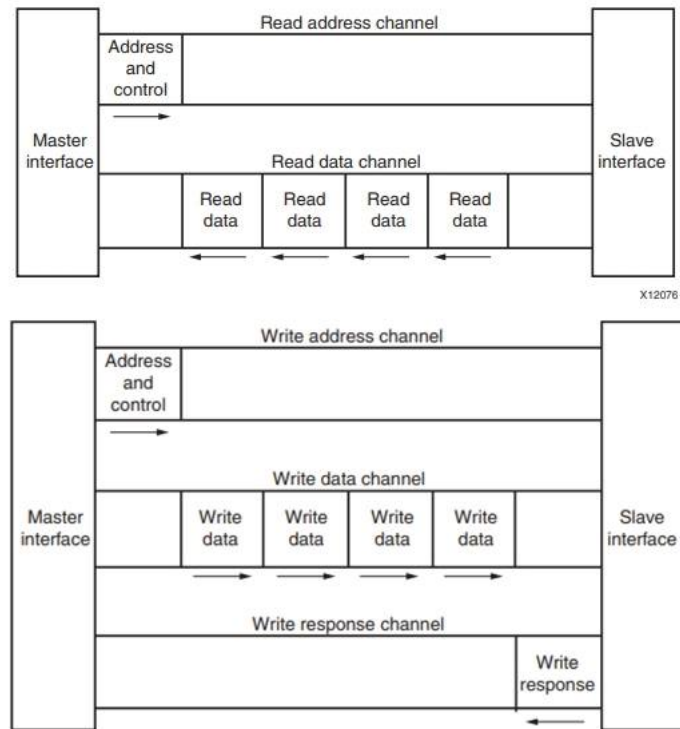


Figura 16 – Fluxo de dados para leitura e escrita no barramento AXI Fone: (XILINX, 2011)

Para realizar escrita e leitura de dados nos IPs através da aplicação implementada em software, utilizamos a função `mmap()` que faz parte da biblioteca `<sys/mman.h>`, esta função cria um novo mapeamento de memória no endereço de memória virtual no seu processo de chamada. Esse novo endereço começa no endereço de memória do IP (com AXI), com isso é possível ler e escrever no endereço memória de qualquer IP, desde que a memória esteja devidamente mapeada.

Como os IPs desenvolvidos por (MUÑOZ et al., 2010) realizam operações em ponto flutuante de 27 bits e o barramento *AXI-4Lite* possui como configuração mínima de registradores o valor de 32 bits, em todos os IPs desenvolvidos as entradas e saídas possuem tamanho de 32 bits, porém para realização dos cálculo é considerada apenas os 27 bits mais significativos presentes nos registradores do barramento de comunicação *AXI-4Lite*.



## 4 Resultados

Neste capítulo é apresentado os resultados obtidos nas implementações do algoritmo de treinamento do classificador LDA nas plataformas *Matlab* (desenvolvido por (LOTTE; GUAN, 2011)), Linguagem C (executado em um SO Linux compilado em um processador *ARM Cortex A9*) e sistema coprocessado hardware-software executado no SoC embarcado no kit de desenvolvimento *Xilinx Zybo Board*.

### 4.1 Resultados obtidos em MATLAB

Em 2010 fabien lotte publicou um trabalho, demonstrando a classificação de sinais de EEG com LDA (LOTTE; GUAN, 2011), usando o extrator de características *common spatial pattenrs*(CSP) e outras variações deste algoritmo. As implementações foram codificadas em *Matlab*, e o autor chegou aos seguintes resultados apresentados na Tabela 4.

Tabela 4 – Acurácia da classificação usando o extrator CSP, classificador LDA e *Data-Set IVa - BCI Competition III*.

BCI III - data set IVa					
Sujeito	<i>aa</i>	<i>al</i>	<i>av</i>	<i>aw</i>	<i>ay</i>
Acurácia(%)	66.07	96.43	47.45	71.88	49.6

Reproduzir os resultados do autor, é de suma importância para este trabalho, tanto para comparação quanto para o entendimento do algoritmo. Utilizando-se do *data set BCI Competition III - IVa*, obteve-se os mesmos resultados obtidos por (LOTTE; GUAN, 2011) apresentados na tabela 4.

Uma outra aplicação do algoritmo é a representação gráfica dos potenciais obtidos em cada um dos 118 sensores posicionados na superfície do crânio do sujeito, apresentado na Figura 17.

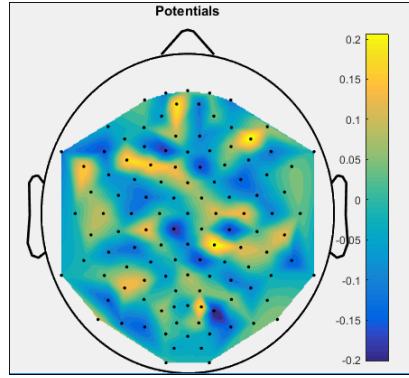


Figura 17 – Resultados obtidos com CSP-LDA

Após execução do algoritmo um dos resultados apresentados na interface gráfica da ferramenta *Matlab* é o tempo de execução do algoritmo de treinamento, onde obteve-se um resultado de tempo médio de processamento de aproximadamente **10.89 ms** (milissegundos).

## 4.2 Resultados Obtidos em Linguagem C

Após a implementação do algoritmo em linguagem C e sua respectiva execução sob o auxílio do sistema operacional Linux, distribuição *Debian*, embarcado nos processadores ARM do kit de desenvolvimento *Zybo Board*. Os resultados obtidos de acurácia de classificação são apresentados na Tabela 5.

Tabela 5 – Acurácia da classificação dos sinais com treinamento em linguagem C.

Sistema	<i>aa</i>	<i>al</i>	<i>av</i>	<i>aw</i>	<i>ay</i>
Linguagem C	65.18%	96.43%	50.51%	59.82%	57.14%

Em comparação com os resultados obtidos na implementação em *Matlab* (Tabela 17), pode-se observar que houveram divergências. Isto ocorreu devido as diferentes arquiteturas de execução, sendo a execução em *Matlab* em arquitetura de tamanho de palavra de 64 bits, enquanto a execução em linguagem C em arquitetura de 32 bits, acarretando em erros de truncamento de palavra, sendo estes propagados durante a execução dos cálculos. Porém, apesar do erro associado, obteve-se uma melhora nos hiperplanos para classificação dos sinais dos sujeitos *ay* e *av*.

No que diz respeito a desempenho computacional esta implementação obteve os melhores resultados, a Tabela 6 apresenta o resultado da análise de perfil.

De acordo com a Tabela 6 a função de covariância é a que mais consome tempo de execução e esforço computacional para realizar o treinamento do classificador, consumindo um total de 81.25% do tempo de execução do algoritmo de treinamento e a



Tabela 6 – Análise de perfil da função de treinamento.

Implementação	#Chamadas	Tempo por Chamada	Tempo Total	%
<i>Cov</i>	2	260 $\mu s$	520 $\mu s$	81.25 $\mu s$
<i>Média</i>	2	40 $\mu s$	80 $\mu s$	12.50 $\mu s$
<i>Sigma</i>	1	40 $\mu s$	40 $\mu s$	6.25 $\mu s$

função média a segunda com maior consumo, registrando um total de 12.5%, enquanto a função sigma consome apenas 6.25%.

Analizando os tempos de execução das funções e seus respectivos números de chamada dentro do algoritmo de treinamento tem-se que o tempo total de execução da função é de 640  $\mu s$ .

## 4.3 Resultados Obtidos no Coprocessamento Hardware-Software

As funções do algoritmo de treinamento do classificador LDA que continuaram mapeadas em software continuaram com os mesmos resultados obtidos na seção anterior. Portanto, esta seção se restringe aos resultados obtidos quanto a implementação das funções em hardware FPGA.

### 4.3.1 Resultados Obtidos Pós Implementação em Hardware

#### 4.3.1.1 IP Para Cálculo de Média

Após a execução de todos os procedimentos de desenvolvimento, síntese e implementação do IP para cálculo de média, com a função de *Report post-Synthesis* da ferramenta *Vivado 2017.4* obteve-se os seguintes estimativas de consumo de hardware, apresentados na Tabela 7.

Tabela 7 – Consumo de hardware da função Média.

	LUT	FF	BUFG	DSP
<i>Média</i>	40%	6%	3%	1%

A maior parte dos recursos consumidos se destinam ao somadores. Para garantir uma maior velocidade de tratamento dos dados, optou-se por não armazenar os sinais de entrada da função média em BRAMs, mas sim, lê-los diretamente dos registradores do barramento *AXI-4Lite*, pois o processo de leitura e escrita nos blocos de memória RAM consomem 4 ciclos de relógio por operação. Além disso, para comando de leitura e escrita seria necessária a implementação de uma FSM o que demandaria mais recursos. Os valores em porcentagem representam o consumo de acordo com as quantidades de

recursos disponíveis no SoC *Zynq* embarcado no kit de desenvolvimento *Zybo Board 70-10* composto por uma FPGA da família *Artix 7*.

#### 4.3.1.2 IP Para Cálculo da Covariância

Utilizando-se da mesma função de ferramenta *Report post-Synthesis*, obteve-se os seguintes resultados de consumo para cálculo da covariância, apresentados na Tabela 8.

Tabela 8 – Consumo de hardware da função Covariância.

	<b>LUT</b>	<b>FF</b>	<b>BUFG</b>	<b>DSP</b>
<i>Covariância</i>	26%	5%	3%	8%

Grande parte do consumo de recursos da função é destinada ao processo de multiplicação e sinais de registradores para conectar os IPs de soma e multiplicação desenvolvidos por (MUÑOZ et al., 2010).

#### 4.3.1.3 Sistema Coprocessado

Após a integralização dos blocos IPs desenvolvidos com o SoC *Zynq* e realizar a implementação do sistema geral (Anexo 01), obteve-se os seguintes resultados para consumo de recursos, apresentados na Tabela 9

Tabela 9 – Consumo de hardware do coprocessamento após implementação.

<b>Recursos</b>	<b>Utilização</b>	<b>Disponível</b>	<b>Utilização (%)</b>
LUT	15252	17600	86.66
LUTRAM	234	6000	3.90
FF	10303	35200	29.27
BRAM	6	60	10.00
DSP	9	80	11.25
IO	36	100	36.00
BUFG	4	32	12.00
MMCM	2	2	100.00

Apesar de ainda haver insumos de hardware disponíveis optou-se por finalizar a implementação seguindo os resultados da Tabela 9. Pois, um consumo de LUT acima de 90% poderia resultar num mal roteamento do circuito.

A Figura 18 e a Figura 19 apresentam o roteamento do sistema de coprocessamento implementado no kit de desenvolvimento e sua legenda, respectivamente.

Os blocos em vermelho representam a alocação dos módulos pertinentes a função de cálculo de média, já se percebe que é a função que mais consumiu recursos. Em azul escuro a alocação dos módulos pertinentes a função de cálculo da covariância. Em azul turquesa a alocação dos módulos do processador *ARM Cortex A9*. Em rosa os módulos

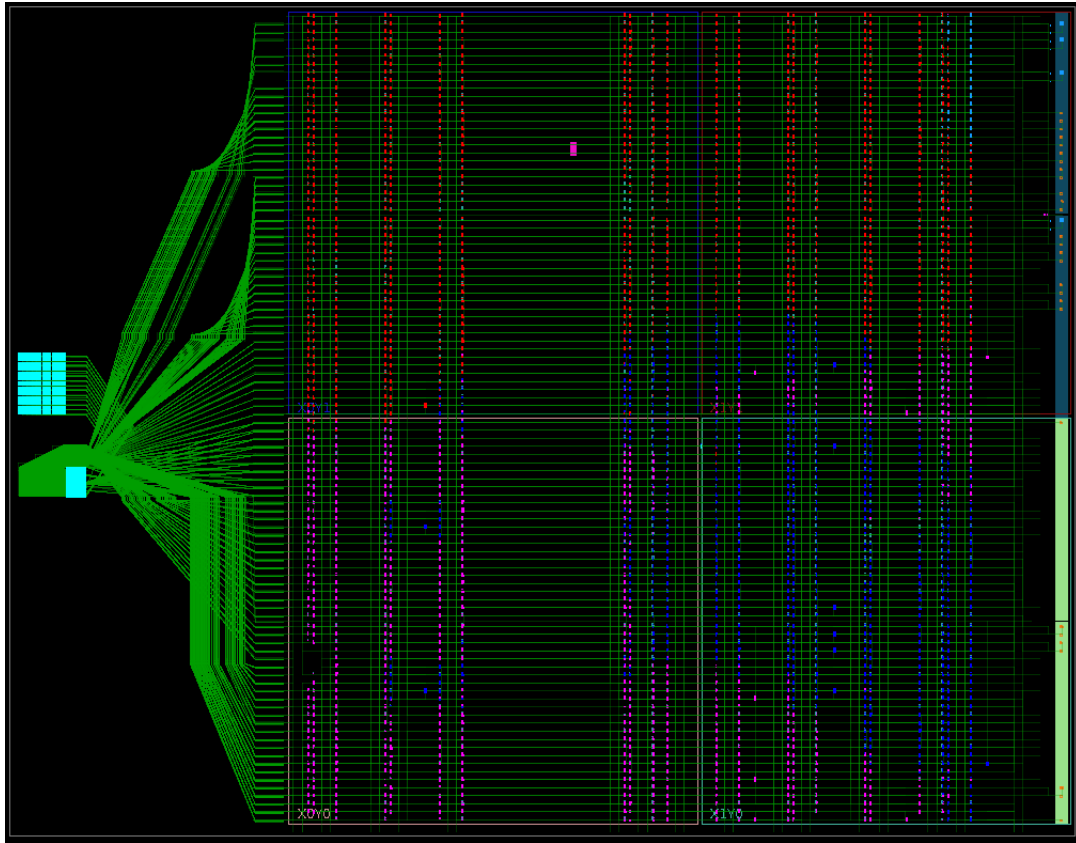


Figura 18 – Roteamento do circuito implementado referente a todo o sistema de coprocessamento.

```

> axi_dispctrl_0 (system_axi_dispctrl_0_0)
> axi_dispctrl_1 (system_axi_dispctrl_1_1)
> axi_i2s_adi_1 (system_axi_i2s_adi_1_0)
> axi_mem_intercon (system_axi_mem_intercon_0)
> axi_protocol_converter_0 (system_axi_protocol_converter_0_0)
> axi_vdma_0 (system_axi_vdma_0_0)
> axi_vdma_1 (system_axi_vdma_1_1)
✓ FP_Mean_Cov_0 (system_FP_Mean_Cov_0_0)
  > Nets (94)
  > U0 (FP_Mean_Cov_v1_0)
    > Nets (94)
    > FP_Mean_Cov_v1_0_S00_AXI_inst (FP_Mean_Cov_v1_0_S00_AXI)
      > Nets (3997)
      > Leaf Cells (2990)
      > Cov0 (Equacao_Cov)
      > Cov1 (Equacao_Cov_190)
      > Cov2 (Equacao_Cov_191)
      > Cov3 (Equacao_Cov_192)
      > divisao (divNR)
      > res_somatorio (FP_Sum_4n)
      > soma1 (FP_Sum_4n_193)
      > soma2 (FP_Sum_4n_194)
      > soma3 (FP_Sum_4n_195)
      > soma4 (FP_Sum_4n_196)
      > soma5 (FP_Sum_4n_197)
      > soma6 (FP_Sum_4n_198)
      > somatorio (addsubfsm_v6)
    > ground (system_ground_0)
    > hdmi_tx_0 (system_hdmi_tx_0_0)
    > processing_system7_0 (system_processing_system7_0_0)
    > processing_system7_0_axi_periph (system_processing_system7_0_axi_periph_2)
    > vdd (system_vdd_1)
    > xconstant_0 (system_xconstant_0_2)

```

Figura 19 – Legenda para a Figura 18.

que compõem o barramento *AXI-4Lite*.

Uma outra informação relevante extraída pós implementação é a estimativa do consumo

energético de todo o sistema desenvolvido. Estas informações são fornecidas pela função *report power* do *Vivado*. A estimativa do consumo energético é apresentado na Figura 20.

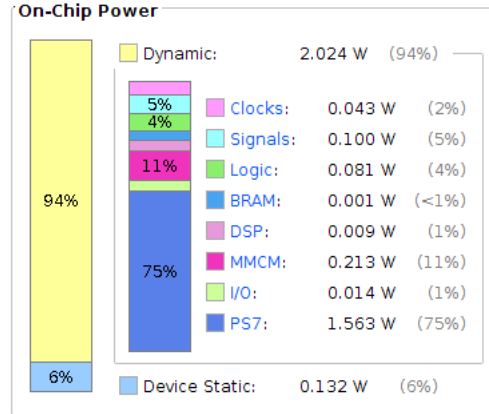


Figura 20 – Estimativa do consumo energético do sistema.

A Figura 20 apresenta a estimativa de consumo energético para cada componente do sistema. O processador ARM representado por PS7 na Figura 20, é o que tem maior consumo, com aproximadamente 75% do consumo total. O sistema em geral apresentou o consumo geral de 2.156 watts.

### 4.3.2 Resultados de Simulação de Hardware

As simulações dos IPs desenvolvidos nos dá uma estimativa de desempenho de processamento dos IPs, com uma análise temporal.

#### 4.3.2.1 IP Para Cálculo de Média

Na plataforma *Vivado 2017.4* e com um *clock* estipulado em 100 MHz, a Figura 21 apresenta a simulação temporal do IP de soma para 20 sinais de entrada.

Na simulação (Figura 21), 20 valores são carregados para a entrada do somador, representados como oP1, oP2,...,oP20. O sinal de *start* realiza o início das operações. A partir da Figura 21 tem-se que o sinal de *start* recebe nível alto no intervalo de tempo  $t = 50ns$ , iniciando o processo de soma dos valores de entrada. O resultado de saída é atualizado no instante de tempo  $t = 145ns$ . Ou seja, do instante de tempo que o *start* é acionado ao instante em que a saída é atualizada tem-se um total de  $95ns$ , isso configura o valor de *latência* do hardware, ou seja, com uma **latência = 95 ns**, a cada sinal de *start* em 95 ns é realizada uma soma de 20 operandos. Por se tratar de uma implementação em *dataflow* o *throughput* é igual a latência do hardware.

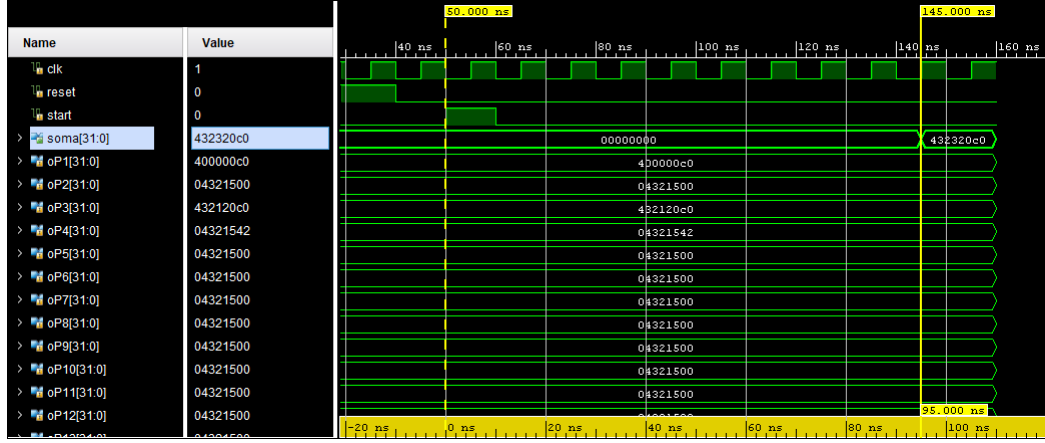


Figura 21 – Simulação do IP de soma de 20 operandos.

Como os dados de treinamento do *dataset - BCI Competition III - IVa* são matrizes de tamanho 80x6 (após a extração de características e a separação das classes), por classe, é necessária a execução do cálculo 4 vezes para somar todos os 80 dados de uma coluna de dados de uma determinada classe. O processamento em hardware retorna então para o processador ARM 1 (um) valor de soma de 20 (vinte) operandos. Como são necessária 4 somas de 20 operandos para totalizar a soma dos 80 (oitenta) amostras, os 4 (quatro) valores retornado para o ARM são retornados via AXI-Lite para o processamento em hardware, onde são somados e em seguida dividido pelo numero de amostras, resultando na média final. Como uma única soma possui  $throughput = 95$  ns, o somador de 4 entradas possui  $throughput = 20$  ns e a divisão tem  $throughput = 30$  ns (obtido também através da simulação do IP), pode-se obter o  $throughput$  para cálculo da média de uma classe de acordo com a Equação 4.1.

$$T_m = N(T_{s20} * 4 + T_{s4} + T_{div}) \quad (4.1)$$

Sendo,  $T_m$  = período para cálculo da média;  $N$  = número de colunas da matriz de características;  $T_{s20}$  =  $throughput$  da soma de 20 operandos;  $T_{s4}$  =  $throughput$  da soma de 4 operandos;  $T_{div}$  =  $throughput$  da divisão. Portanto para o IP de média obtemos um  $T_m = 2.58 \mu s$ . Ou seja, para calcular a média de cada classe será necessário um período de  $2.58 \mu s$ . Como são duas classes, o período total de processamento utilizado pela função média é de **5.16  $\mu s$** .

#### 4.3.2.2 IP Para Cálculo de Covariância

Na mesma ferramenta e mesmo *clock* utilizado na simulação do IP de cálculo de média, realizou-se a simulação do IP de covariância. A Figura 22 apresenta a simulação temporal do IP que realiza o cálculo da Equação 3.1.

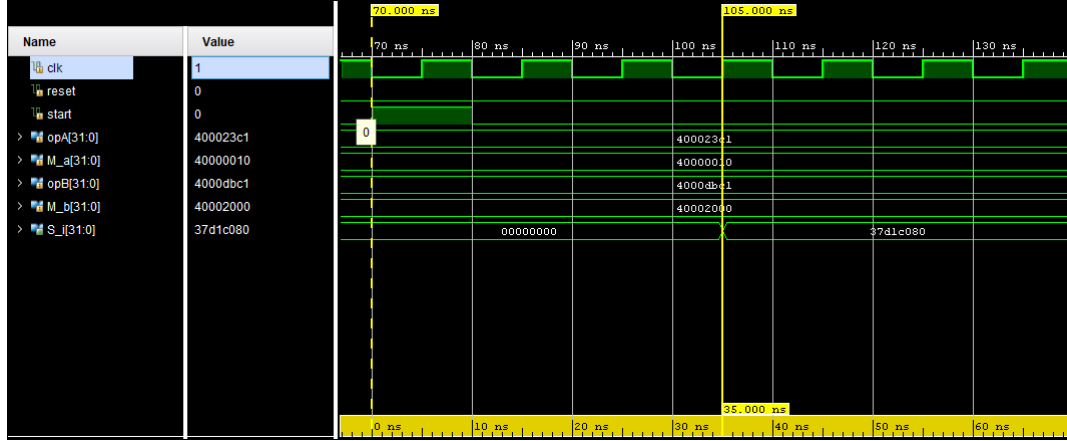


Figura 22 – Simulação do IP de covariância para um único módulo.

De acordo com a Figura 22 os quatro valores necessários para o cálculo são carregados nos quatro operadores de entrada. Com o *start* ativado em  $t = 70ns$  o primeiro resultado de saída é obtido em  $t = 105ns$ , portanto a latência do hardware é de  $t = 35ns$ , como a implementação também segue uma técnica de *dataflow* este valor também é referido ao *throughput* do hardware. Como são 4 implementações executando em paralelo então a cada 35 ns partindo do *start* obtém-se 4 valores de saídas referente a Equação 3.1. Como os dados de entrada se referem a uma matriz de tamanho 80x6 para cada classe, pode-se estimar o *throughput* do hardware desenvolvido, a partir da Equação 4.2.

$$T_{cov} = \frac{N_{class}(N_{col}^2)(T_{cov1})}{N_{par}} \quad (4.2)$$

Sendo,  $T_{cov}$  = período total de processamento da função;  $N$  = número de colunas da matriz de características;  $N_{class}$  = número de classes;  $N_{col}$  = número de colunas da matriz de entrada;  $T_{cov1}$  = *throughput* da covariância,  $N_{par}$  = número de implementações sendo executadas em paralelo.

Sendo assim, pela Equação 4.2 tem-se que  $T_{cov} = 50.4 \mu s$ , ou seja, o IP de cálculo em ponto flutuante da função covariância consome um tempo de processamento de  $50.4 \mu s$  para calcular a covariância das duas classes dos dados de treinamento do *data-set* - *BCI Competition III - IVa*.

### 4.3.3 Resultados de Execução

Após todo o processo de implementação e simulação, executou-se o sistema co-processado utilizando-se do *data-set* - *BCI Competition III - IVa* como sinais de entrada, obtendo-se os seguintes resultados de acurácia apresentados na Tabela 10.

O sistema obteve uma melhor acurácia no sujeito *al*, pois é o sujeito que mais possui tarefas realizadas e previamente classificadas. O oposto ocorre com o sujeito *av*.

Tabela 10 – Acurácia da classificação dos sinais com treinamento em sistema coprocessado.

Sistema	<i>aa</i>	<i>al</i>	<i>av</i>	<i>aw</i>	<i>ay</i>
Coprocessado	65.18%	96.43%	50.51%	59.82%	57.14%

Outro dado importante coletado após a execução foi o tempo de processamento da função de treinamento implementada no sistema coprocessado. Realizando a análise de perfil, obteve-se um tempo de processamento de **10 milissegundos**.

#### 4.3.4 Análise de Resultados de Desempenho de Processamento

Em cada uma das implementações foram utilizadas arquiteturas diferentes, a Tabela 11 apresenta as arquiteturas utilizadas em cada uma das implementações.

Tabela 11 – Propriedades das plataformas utilizadas.

Implementação	Sistema	Processador(es)	Dados
<i>Matlab</i>	Windows 10	Intel Core i5 (2.2 GHz)	64 bits
<i>Linguagem C</i>	Linux	ARM Cortex A9 (650 MHz)	32 bits
<i>Hardware-Software</i>	<i>Zynq</i>	ARM /Artix 7 (100 MHz)	27 bits

A partir da execução de cada uma das implementações obteve-se os seguintes resultados de acurácia apresentados na Tabela 12.

Tabela 12 – Acurácia das implementações de estudo.

Implementação	<i>aa</i>	<i>al</i>	<i>av</i>	<i>aw</i>	<i>ay</i>
<i>Matlab</i>	66.70%	96.43%	47.75%	71.88%	49.60%
<i>Linguagem C</i>	66.18%	96.43%	50.51%	59.82%	57.14%
<i>Hardware-Software</i>	66.18%	96.43%	50.51%	59.82%	57.14%

De acordo com a Tabela 12, as implementações em linguagem C e coprocessamento obtiveram valores diferentes da implementação original em *Matlab*, devido ao que apresentado na Tabela 11, apresentam tamanho de dados inferiores à implementação em *Matlab*. Com isso todos os sinais foram truncados em seus respectivos tamanhos de dados, propagando erros em seus cálculos em ponto flutuante.

Além disso a implementação em sistema coprocessado apresentou um erro quadrático médio de 0,42% em relação à implementação *Matlab*.

Por se tratarem de sistemas com frequências de *clock* diferentes cada uma das implementações apresenta tempos de execução diferentes. Estes tempos são apresentados na Tabela 13.

A implementação que obteve o melhor desempenho de processamento foi a implementação em Linguagem C embarcado no processador *ARM Cortex A9* presente no

Tabela 13 – Tempos de execução da função de treinamento nas diferentes implementações.

Implementação	Tempo de execução da função de treinamento
<i>Matlab</i>	10.89 ms
<i>Linguagem C</i>	640 $\mu$ s
<i>Hardware-Software</i>	10 ms

*SoC* do kit de desenvolvimento *Zybo Board*. A implementação com menor desempenho de tempo de processamento foi a implementação em *Matlab*, isto é devido o fato da implementação estar sendo executada em uma plataforma *Windows 10* e por se tratar de uma linguagem de programação interpretada os outros processos executados juntos com a execução do algoritmo interfere em seu processamento. Já a respeito da implementação em sistema coprocessado o que se esperava era um melhor desempenho pois de acordo com os resultados obtidos em simulação os IPs de cálculos de média e de covariância apresentaram um tempo de execução extremamente baixo quando comparado com o resultado de execução obtido.



## 5 Conclusão

Este trabalho apresentou um estudo da implementação em sistema coprocessado hardware-software do algoritmo de treinamento do classificador LDA, comparando os resultados obtidos com outras duas implementações em arquiteturas diferentes, sendo elas a implementação em *Matlab* (Desenvolvido por Fabien Lotte) e a implementação em linguagem C.

De acordo com os resultados obtidos em simulação a expectativa era de se obter um menor tempo de processamento na implementação coprocessada, entretanto nos resultados simulados não se fez presente o tempo de tráfego de dados no barramento *AXI-4Lite*, entre o processador (ARM) e a lógica programável (IPs). Com isso a partir dos resultados da Tabela 13 é possível concluir que o sistema coprocessado não é o mais eficiente em tempo de execução, ficando atrás da implementação em linguagem C, mas ainda assim mais eficiente que a implementação em *Matlab*. Isso pode ser decorrente do ambiente de execução da implementação em C, que foi executada sobre o auxílio de um sistema operacional Linux (versão para desenvolvedores), sendo esta com uma quantidade de processos reduzidas, conseqüentemente consome menos recurso de processamento. Sendo assim o objetivo deste presente trabalho foi alcançado, mesmo apresentando resultados inesperados.

Notou-se também que em termos computacionais o algoritmo de treinamento do classificador LDA, não necessita de um grande esforço computacional, como pôde ser observado na Tabela 13, isso mostra que o algoritmo pode ser implementado em outras arquiteturas de hardware, por exemplo kits de desenvolvimento como *Raspberry-Pi*, *Beagle-Bone* entre outros.

Para o desenvolvimento deste trabalho a maior dificuldade encontrada foi embarcar o sistema operacional Linux nos cores ARM do kit utilizado. Esta tarefa foi a que mais demandou tempo de trabalho e pesquisa, pois ambos os autores deste trabalho não eram munidos do conhecimento para tal atividade.

Este trabalho mostrou que é possível aplicar métodos de processamento em arquitetura de hardware reconfigurável aplicado à classificação de sinais, portanto para trabalhos futuros, pode ser feito uma melhor interface entre hardware e software para obter um melhor tempo de execução, pois os resultados deste trabalho mostra que o barramento

*AXI-4Lite* foi um limitante para se ter os resultados esperados.

# Referências

- ALKIM, E.; KILIÇ, E. Chip design for intelligent data classification algorithms and implementation on an fpga: A case study to classify emg signals. In: *2011 IEEE 19th Signal Processing and Communications Applications Conference (SIU)*. [S.l.: s.n.], 2011. p. 307–310. ISSN 2165-0608. Citado na página 42.
- ALVAREZ, A.; LEMOS, I. d. C. Os neurobiomecanismos do aprender: a aplicação de novos conceitos no dia-a-dia escolar e terapêutico. *Revista Psicopedagogia*, Associação Brasileira de Psicopedagogia, v. 23, n. 71, p. 181–190, 2006. Citado 3 vezes nas páginas 29, 30 e 31.
- BLANKERTZ, B. et al. *BCI Competition III*. 2005. Disponível em: <[http://www.bbci.de/competition/iii/desc\\_IVa.html](http://www.bbci.de/competition/iii/desc_IVa.html)>. Citado 2 vezes nas páginas 36 e 37.
- BLANKERTZ, B. et al. The bci competition iii: validating alternative approaches to actual bci problems. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, v. 14, n. 2, p. 153–159, June 2006. ISSN 1534-4320. Citado 4 vezes nas páginas 17, 27, 35 e 36.
- BRUNELLI, R. *Template matching techniques in computer vision: theory and practice*. [S.l.]: John Wiley & Sons, 2009. Citado na página 25.
- CAMPISI, P.; ROCCA, D. L.; SCARANO, G. Eeg for automatic person recognition. *Computer, IEEE*, v. 45, n. 7, p. 87–89, 2012. Citado 3 vezes nas páginas 15, 32 e 33.
- CAO, Y. et al. A post-silicon trace analysis approach for system-on-chip protocol debug. In: IEEE. *2017 IEEE 35th International Conference on Computer Design (ICCD)*. [S.l.], 2017. p. 177–184. Citado 2 vezes nas páginas 15 e 39.
- CHEN, X. et al. Soc-based architecture for robotic prosthetics control using surface electromyography. In: *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*. [S.l.: s.n.], 2016. v. 01, p. 134–137. Citado na página 42.
- CLARK NASHAAT BOUTROS, M. M. D. *The Brain and Behavior: An Introduction to Behavioral Neuroanatomy*. 2. ed. [S.l.]: Cambridge University Press, 2005. ISBN 0521840503, 9780521840507. Citado na página 29.
- CROCKETT, L. H. et al. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. [S.l.]: Strathclyde Academic Media, 2014. Citado 6 vezes nas páginas 15, 26, 27, 38, 39 e 49.
- DIGILENT. *Zybo Board Reference Manual*. 2017. Disponível em: <<https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual>>. Citado 3 vezes nas páginas 15, 39 e 40.
- ELECTRICAL, I. of; ENGINEERS, E. *IEEE Std 754-2008 - IEEE Standard for Floating-Point Arithmetic*. 2008. Disponível em: <<https://standards.ieee.org/findstds/standard/754-2008.html>>. Citado na página 45.

GLETTE; TORRESEN. An online ehw pattern recognition system applied to sonar spectrum classification. *Evolvable Systems: From Biology to Hardware*, Springer, p. 1–12, 2007. Citado na página 42.

GLETTE, K.; TORRESEN, J. Intermediate level fpga reconfiguration for an online ehw pattern recognition system. In: *2009 NASA/ESA Conference on Adaptive Hardware and Systems*. [S.l.: s.n.], 2009. p. 19–26. Citado na página 42.

GUGER, C. et al. *Brain-Computer Interface Research: A State-of-the-Art Summary*. 1. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 2013. (SpringerBriefs in Electrical and Computer Engineering). ISBN 978-3-642-36082-4, 978-3-642-36083-1. Citado 2 vezes nas páginas 33 e 34.

IRICK, K. et al. A hardware efficient support vector machine architecture for fpga. In: *2008 16th International Symposium on Field-Programmable Custom Computing Machines*. [S.l.: s.n.], 2008. p. 304–305. Citado na página 42.

IZENMAN, A. J. *Modern multivariate statistical techniques*. [S.l.]: Springer, 2008. v. 1. Citado na página 37.

KAIS, B. et al. An embedded implementation of home devices control system based on brain computer interface. In: *2014 26th International Conference on Microelectronics (ICM)*. [S.l.: s.n.], 2014. p. 140–143. ISSN 2159-1660. Citado na página 42.

KANDEL, e. a. E. R. *Principles of Neural Science*. 5. ed. The address: Mc Graw Hill, 2013. v. 2. An optional note. ISBN 978007181001-2. Citado 4 vezes nas páginas 15, 25, 29 e 30.

KETSUWAN, R.; PADUNGWEANG, P. A linear discriminant analysis using weighted local structure information. In: IEEE. *Computer Science and Software Engineering (JCSSE), 2017 14th International Joint Conference on*. [S.l.], 2017. p. 1–5. Citado na página 37.

KORKMAZ, S. A. et al. A expert system for stomach cancer images with artificial neural network by using hog features and linear discriminant analysis: Hog\_lda\_ann. In: IEEE. *Intelligent Systems and Informatics (SISY), 2017 IEEE 15th International Symposium on*. [S.l.], 2017. p. 000327–000332. Citado na página 37.

LOTTE, F. *Study of electroencephalographic signal processing and classification techniques towards the use of brain-computer interfaces in virtual reality applications*. Tese (Doutorado) — INSA de Rennes, 2008. Citado na página 25.

LOTTE, F.; GUAN, C. Regularizing common spatial patterns to improve bci designs: Unified theory and new algorithms. *IEEE Transactions on Biomedical Engineering*, v. 58, n. 2, p. 355–362, Feb 2011. ISSN 0018-9294. Citado 12 vezes nas páginas 15, 17, 25, 26, 27, 33, 42, 43, 44, 45, 50 e 53.

MASON, S. G.; BIRCH, G. E. A general framework for brain-computer interface design. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, v. 11, n. 1, p. 70–85, March 2003. ISSN 1534-4320. Citado 3 vezes nas páginas 25, 27 e 35.

MUÑOZ, D. M. et al. Tradeoff of fpga design of a floating-point library for arithmetic operators. *Journal of Integrated Circuits and Systems*, v. 5, n. 1, p. 42–52, 2010. Citado 4 vezes nas páginas 45, 46, 51 e 56.

- RAO, R. P. *Brain-computer interfacing: an introduction*. [S.l.]: Cambridge University Press, 2013. Citado 2 vezes nas páginas 31 e 32.
- SHASHOA, N. A. A. et al. Classification depend on linear discriminant analysis using desired outputs. In: IEEE. *Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2016 17th International Conference on. [S.l.], 2016. p. 328–332. Citado na página 26.
- SINGH, A.; PRAKASH, B. S.; CHANDRASEKARAN, K. A comparison of linear discriminant analysis and ridge classifier on twitter data. In: IEEE. *Computing, Communication and Automation (ICCCA)*, 2016 International Conference on. [S.l.], 2016. p. 133–138. Citado na página 37.
- SIQUEIRA, T. M. *Implementação de um Classificador de Imagens Baseado em Redes Neurais em Sistemas Embarcados*. Dissertação (Mestrado) — Universidade de Brasília, 7 2016. Citado na página 48.
- SIULY, S. *Analysis and Classification of EEG Signals*. Dissertação (Mestrado) — University of Southern Queensland, <https://www.springer.com/gp/book/9783319476520>, 7 2012. Citado 8 vezes nas páginas 15, 25, 30, 31, 32, 33, 34 e 35.
- SIULY, S.; LI, Y.; ZHANG, Y. *EEG Signal Analysis and Classification: Techniques and Applications*. [S.l.]: Springer, 2017. Citado 4 vezes nas páginas 25, 26, 31 e 32.
- TEPLAN, M. et al. Fundamentals of eeg measurement. *Measurement science review*, v. 2, n. 2, p. 1–11, 2002. Citado na página 30.
- THEODORIDIS, S.; KOUTROUMBAS, K. et al. *Pattern recognition*. [S.l.]: Academic press London, 1999. Citado na página 26.
- WANG, Y. et al. Bci competition 2003-data set iv:an algorithm based on cssd and fda for classifying single-trial eeg. *IEEE Transactions on Biomedical Engineering*, v. 51, n. 6, p. 1081–1086, June 2004. ISSN 0018-9294. Citado na página 41.
- WOLPAW, E. W. W. J. R. *Brain-Computer Interfaces: Principles and Practice*. Oxford University Press, 2012. ISBN 9780195388855 0195388852. Disponível em: <<http://gen.lib.rus.ec/book/index.php?md5=46D2894771F9769A0D5B429B681CBF33>>. Citado 3 vezes nas páginas 15, 33 e 34.
- XILINX. *AXI Reference Guide*. 2011. Disponível em: <[https://www.xilinx.com/support/documentation/ip\\_documentation/ug761\\_axi\\_reference\\_guide.pdf](https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf)>. Citado 3 vezes nas páginas 15, 40 e 51.
- YANG, F. Implementation of an rbf neural network on embedded systems: real-time face tracking and identity verification. *IEEE Transactions on Neural Networks*, v. 14, n. 5, p. 1162–1175, Sept 2003. ISSN 1045-9227. Citado na página 42.
- YUAN, Y. et al. Comparison of gpu and fpga based hardware platforms for ultrasonic flaw detection using support vector machines. In: *2017 IEEE International Ultrasonics Symposium (IUS)*. [S.l.: s.n.], 2017. p. 1–4. Citado na página 42.



# Anexos





# ANEXO A – Sistema Geral

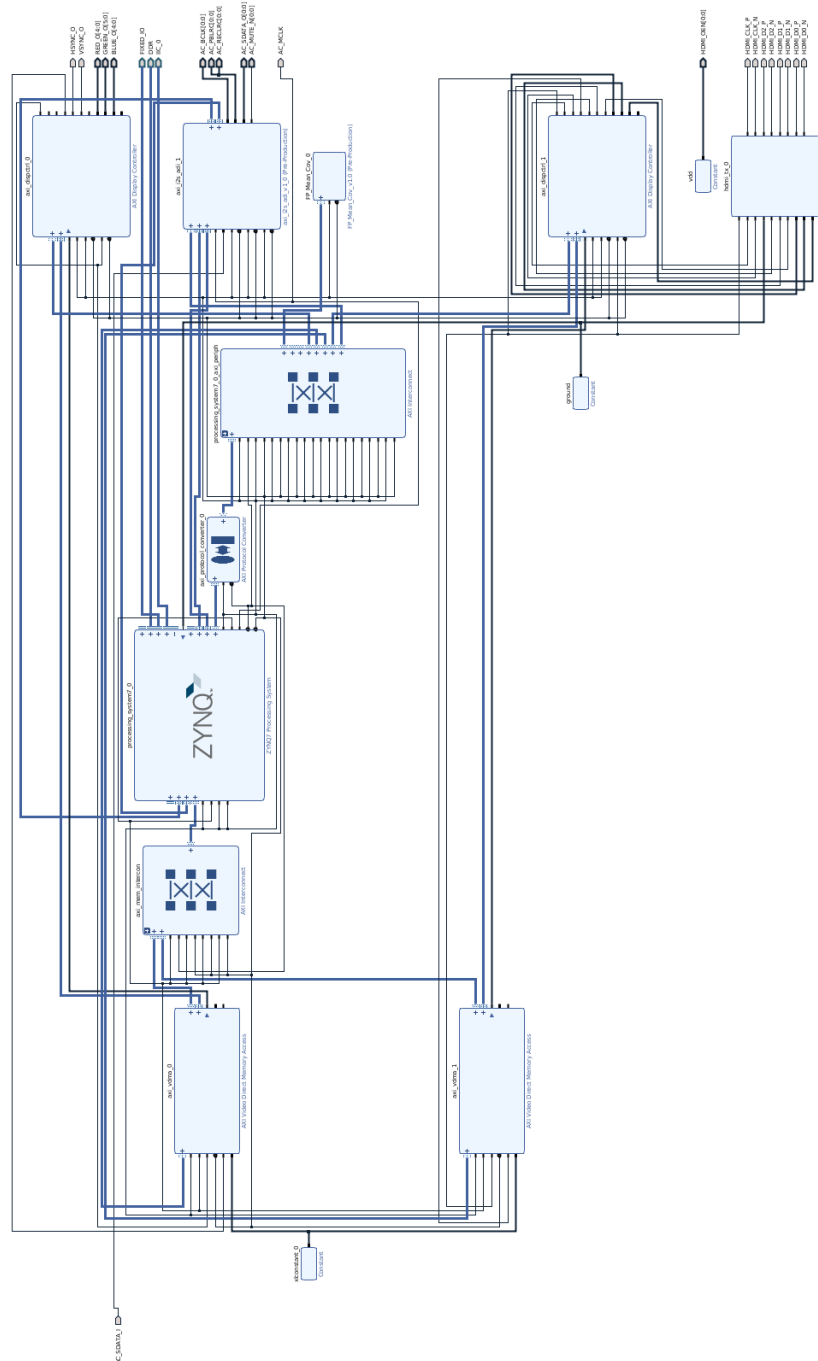


Figura 23 – Sistema geral com IP



# ANEXO B – Tutorial para Instalação do Sistema Operacional Linux no Zybo

Este tutorial foi elaborado para ambiente Linux e baseia-se no tutorial disponibilizado por (??).

## B.0.1 Geração do Arquivo u-boot.elf

### B.0.1.1 Requisitos Necessários

- Vivado 2014.1 WebPACK ou superior, disponível em : <<https://www.xilinx.com/support/download.html>>
- Acesso ao repositório no git para "u-boot-xlnx" disponível em : <<https://github.com/Xilinx/u-boot-xlnx>>

### B.0.1.2 Instruções

Para começarmos precisamos baixar o repositório do u-boot, para isso usamos o comando:

```
$git clone https://github.com/Xilinx/u-boot-xlnx.git
```

Este comando faz o download do repositório u-boot-xlnx mantendo pela Xilinx, em seguida entramos no diretório:

```
$cd u-boot-xlnx/
```

Dentro do diretório, é necessário carregar as ferramentas de compilação, vamos usar o comando:

```
$source /opt/Xilinx/Vivado/2017.4/settings64.sh
```

Neste ponto iremos limpar todos os arquivos já compilados que podem estar no repositório, para isso usamos os seguintes comandos:

```
$make mrproper
$make clean
```

Agora vamos adicionar as variáveis de ambiente para compilação e o arquivo de configuração para o kit Zybo com os comandos abaixo:

```
$make CROSS_COMPILE=arm-linux-gnueabi- zynq_zybo_config
```

o arquivo de configuração foi criado, para compilar usamos o comando:

```
$make CROSS_COMPILE=arm-linux-gnueabihf-
```

Por fim vamos adicionar a extensão ".elf" ao arquivo "u-boot", usaremos o seguinte comando:

```
$mv u-boot u-boot.elf
```

Esta etapa foi concluída, vamos agora criar o arquivo BOOT.bin.

## B.0.2 Geração do Arquivo BOOT.bin

Requisitos Necessários:

- Vivado 2014.1 WebPACK ou superior, disponível em : <<https://www.xilinx.com/support/download.html>>
- ZYBO Base System Design disponível em: <[https://reference.digilentinc.com/\\_media/reference/programmable-logic/zybo/zybo\\_base\\_system.zip](https://reference.digilentinc.com/_media/reference/programmable-logic/zybo/zybo_base_system.zip)>
- completar o passo anterior

### B.0.2.1 Instruções

Para criarmos o arquivo BOOT.bin primeiramente descompactamos o arquivo "zybo\_base\_system.zip" podemos usar o seguinte comando:

```
$unzip zybo_base_system.zip
```

Após descompactar iremos até o diretório:

```
$zybo_base_system/source/vivado/hw/zybo_bsd/
```

Com o software vivado iremos abrir o arquivo "zybo\_bsd.xpr". No vivado executaremos a função: **Generate Bitstream**.

*obs: Esse passo irá demorar alguns minutos.*

Agora exportaremos o ".bit" para a pasta de projeto, no Vivado seguimos os passos: **File > Export > Export Hardware** na janela que surgirá, marcaremos a opção **Include Bitstream**. Ainda no Vivado fazemos os passos: **File > Launch SDK**

*obs: Todos os passos agora serão executados do software SDK*

No SDK vamos criar o arquivo fsbl.elf (first stage boot loader). Este é responsável por carregar o arquivo de configuração (bitstream) no sistema de processamento do Zybo. Para criá-lo vamos seguir os passos: **File > New > Application Project** no campo "Project Name" usaremos "fsbl" e clicamos em **Next**, agora escolheremos o template **Zynq**

**FSBL**, por fim clicamos em **Finish**.

Neste ponto estamos de posse dos arquivos **u-boot.elf**, **fsbl.elf** e **system\_wrapper.bit**, estes são necessários para gerarmos o **BOOT.bin**.

Seguiremos então os passos: **Xilinx > Create Boot Image**, na janela que surgirá escolhemos o diretório de saída de nossa preferência. No campo "Boot Image Partitions", iremos adicionar os seguintes arquivos em ordem; *obs: A ordem dos arquivos é muito importante para esse processo.* em **Add** adicionaremos o arquivo **fsbl.elf** que se encontra na pasta de projeto:

```
zybo_bsd.sdk/fsbl/Debug/fsbl.elf
```

Posteriormente adicionaremos o arquivo **system\_wrapper.bit** que se encontra no diretório:

```
zybo_bsd.sdk/system\wrapper_boards\platform_hw_0/system_wrapper.bit
```

e por fim o arquivo **u-boot.elf**, que se encontra no diretório que clonamos do git:

```
u-boot-xlnx/u-boot.elf
```

Com esses três arquivos adicionados clicamos em **Create Image**. No diretório de saída que escolhemos estará o arquivo **BOOT.bin**.

## B.1 Compilando o Kernel Linux

### B.1.1 Requisitos Necessários

- Vivavo 2014.1 WebPACK ou superior, disponível em : <https://www.xilinx.com/support/download.html>
- ZYBO Base System Design disponível em: [https://reference.digilentinc.com/\\_media/reference/programmable-logic/zybo/zybo\\_base\\_system.zip](https://reference.digilentinc.com/_media/reference/programmable-logic/zybo/zybo_base_system.zip)

#### B.1.1.1 Instruções

Para começarmos precisamos baixar o repositório do linux-xlnx, para isso usamos o comando:

```
$git clone https://github.com/Xilinx/linux-xlnx.git
```

Este comando faz o download do repositório linux-xlnx mantendo pela Xilinx, em seguida entramos no diretório:

```
$cd linux-xlnx/
```

Dentro do diretório, é necessário carregar as ferramentas de compilação, vamos usar o comando:

```
$source /opt/Xilinx/Vivado/2017.4/settings64.sh
```

Neste ponto iremos limpar todos os arquivos já compilados que podem estar no repositório, para isso usamos os seguintes comandos:

```
$make mrproper
```

```
$make clean
```

Agora vamos adicionar as variáveis de ambiente para compilação e o arquivo de configuração para o kit Zybo com os comandos abaixo:

```
$make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- xilinx_zynq_defconfig
```

o arquivo de configuração foi criado, para compilar o kernel usamos o comando:

```
$make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

*obs: Este comando pode demorar alguns minutos para terminar.*