Final

Desenvolvimento e Avaliação de um Sistema de Medição Fisiológica de Baixo Custo

PROJETO DE MESTRADO

Juan Miguel Teixeira Sousa
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES
E REDES DE ENERGIA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

março | 2016

Desenvolvimento e Avaliação de um Sistema de Medição Fisiológica de Baixo Custo

PROJETO DE MESTRADO

Juan Miguel Teixeira Sousa
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES
E REDES DE ENERGIA

ORIENTAÇÃO
Luís Armando de Aguiar Oliveira Gomes

CO-ORIENTAÇÃO
Sergi Bermúdez i Badia

Resumo

Atualmente os sistemas de aquisição de medidas fisiológicas, tais como o eletroencefalograma, o eletrocardiograma e o eletromiograma, são complexos e dispendiosos. Este trabalho visou o desenvolvimento de um protótipo que possa ser considerado um dispositivo de baixo custo, com grande resolução, e que permita a aquisição das medidas fisiológicas referidas anteriormente, num único dispositivo.

Tendo sido estudados os diversos equipamentos disponíveis, optou-se por desenvolver o projeto com base num dispositivo da *Texas Instruments*, o ADS1299, que foi criado com o intuito de servir de andar de entrada em sistemas de instrumentação médica, apresentando 8 canais de medição (expansíveis no máximo a 32, através de multiplexagem de dispositivos), com um custo inferior a 300 €. Este trabalho envolveu a construção do protótipo de aquisição, bem como o desenvolvimento e alteração do código do firmware para o hardware. Diversos testes foram realizados para comparar o desempenho do protótipo desenvolvido face aos dispositivos comerciais de elevado custo e de alto desempenho, disponíveis na Universidade da Madeira. Os resultados obtidos com o protótipo desenvolvido provam que o mesmo pode servir como uma ferramenta de pesquisa, de baixo custo atendendo à relação preço/performance, facilitando a interação cérebro-computador e permitindo criar paradigmas de treino mental do tipo *neuro-feedback* para pessoas com danos cerebrais.

Palavras-chave: Medidas Fisiológicas, eletroencefalograma, eletrocardiograma, eletromiograma, dispositivos fisiológicos de baixo custo, amplificadores de baixo ruído.

Abstract

Currently the systems of acquisition of physiologic measurements, such as electroencephalograms, electrocardiograms and electromyograms, are complex and expensive. This work aimed at developing a low cost prototype with high resolution and allowing the acquisition of the physiologic measurements mentioned above, in a single device.

Having studied several available equipment, it was decided to develop the project based on the ADS1299 from Texas Instruments, which was created with the aim to serve as the input stage of a medical measurement instrument having 8 measuring channels (expandable to a maximum of 32 through multiplexing devices), with a cost of less than € 300. This project involved the construction of the acquisition prototype as well as the development and the alteration of the firmware code for the hardware. Several tests were performed to compare the performance of the developed prototype in relation to high cost and high performance commercial devices available at the University of Madeira. The results obtained with the developed prototype prove that it can serve as a low cost research tool given the price/performance ratio, facilitating the brain-computer interaction and allowing the creating of mental training paradigms of the neuro-feedback for people with brain damage.

Keywords: Physiological Measurements, electroencephalogram, electrocardiogram, electromyography, inexpensive physiological devices, low-noise amplifiers.

Agradecimentos

A realização deste projeto deve-se em muito à colaboração, apoio e incentivo de diversas pessoas, que direta ou indiretamente, estiveram ligadas à elaboração deste projeto final de mestrado; a estas pessoas transmito o meu sincero agradecimento.

À Universidade da Madeira, como Instituição de Ensino Superior, que sempre teve docentes dispostos a ajudar e a ensinar, criando condições favoráveis para a minha formação como mestre, preparando-me assim para o mundo do trabalho nas áreas de Telecomunicações e Redes de Energia.

Aos orientadores do Projeto, Professor. Dr. Luís Aguiar Gomes e Professor Dr. Sergi Bermudez i Badia, e ainda ao aluno de Doutoramento Athanasios Vouvopoulos, que sempre estiveram disponíveis para prestar o apoio necessário tanto no desenvolvimento do projeto, como na elaboração do relatório, sendo sensíveis as dificuldades sentidas no decorrer deste tempo.

Aos Engenheiros Filipe Santos, Carlos Francisco e Pedro Camacho e em especial ao colega de curso Jorge Lopes, por toda a ajuda prestada.

À minha família, namorada e amigos pela compreensão e apoio incondicional que tantas vezes me mostraram que era possível.

Índice

Resumo iii

Abstract v

Agradecimentos vii

Índice ix

Índice de Figuras xiii

Índice de Tabelas xix

Acrónimos xxi

1 Introdução 1

1.1 Contexto e Motivação 1

1.2 Objetivos 1

1.3 Estrutura da Tese 2

2 Medidas Fisiológicas 5

2.1 Variáveis Fisiológicas 5

2.2 Tipos de Medidas Fisiológicas 5

2.3 Eletroencefalograma (EEG) 5

2.3.1 Anatomia do Cérebro Humano 5

2.3.2 Descrição de um EEG 7

2.3.3 Sistema 10-20 e Montagem de Elétrodos 8

2.3.4 Processamento de Sinal do EEG 9

2.3.5 Ritmos do EEG 10

2.3.6 Alterações do EEG 12

2.4 Eletrocardiograma (ECG) 13

2.4.1 Anatomia do Coração Humano 13

2.4.2 Descrição do ECG 14

2.4.3 Potenciais de Superfície e Aquisição de Sinais de ECG 15

2.5 Eletromiograma (EMG) 18

2.6 Conclusões do Capítulo 18

3 Instrumentação Médica e Dispositivos de Medição Fisiológica 19

3.1 Sistemas de Instrumentação Médica 19

3.1.1 Desafios 19

3.1.2 Requisitos 20

3.1.3 Blocos Funcionais 21

3.2 Aplicações 22

3.2.1 Interfaces Cérebro-Computador (BCI) e Aplicações 22

3.3 Dispositivos Comerciais para Medidas Fisiológicas 23

3.3.1 Dispositivo *g.tec g.MOBIlab+* 23

3.3.2 Dispositivo *Enobio NE* 24

3.3.3 Dispositivo *ModularEEG* 25

3.3.4 Dispositivo *PLUX BioSignals* 25

3.3.5 Dispositivo *BITalino BioSignals* 26

3.4 Conclusões do Capítulo 26

4 Dispositivo ADS1299 29
4.1 Analog Front-End 29
4.2 Caraterísticas do ADS1299 30
4.2.1 Características Elétricas 31
4.3 Comunicação com o ADS1299 32
4.3.1 Comunicação SPI 32
4.4 Comandos para o ADS1299 33
4.4.1 Comandos do Sistema 33
4.4.2 Comandos de Aquisição de Dados 33
4.4.3 Comandos de Leitura e de Escrita dos Registos 34
4.5 Programação do ADS1299 34
4.6 Operação do ADS1299 41
4.7 Conclusões do Capítulo 42

5 Protótipo I 43
5.1 Hardware: ADS1299 + DSP MMB0 43
5.2 Software: Aplicação em *LabVIEW* 45
5.2.1 Interface da Aplicação 45
5.3 Configuração dos Registos 48
5.4 Resultados Obtidos 48
5.4.1 Medição do Ruído no Protótipo I 49
5.4.2 Obtenção de Ondas Geradas Internamente pelo ADS1299 com o Protótipo I 49
5.4.3 Obtenção de Ondas Geradas por um Gerador de Sinais com o Protótipo I 50
5.5 Conclusões do Capítulo 53

6 Protótipo II 55
6.1 Hardware 55
6.1.1 Escolha do Microcontrolador 55
6.1.2 Escolha do Dispositivo de Comunicação sem Fios 56
6.2 Alimentação 57
6.3 Comunicação SPI 57
6.4 Construção do Protótipo II 58
6.4.1 Primeira Fase 58
6.4.2 Segunda Fase 58
6.4.3 Terceira Fase 60
6.5 Custo do Protótipo II 64
6.6 Código e Programação do Protótipo II 65
6.6.1 Estrutura do Código 66
6.6.2 Apresentação do Código 66
6.6.3 Modo de funcionamento do dispositivo ADS1299 70
6.6.4 Programação do Protótipo II 70
6.7 Visualização dos Dados do Protótipo II 70
6.7.1 Visualização dos Dados no Software Arduino *IDE* 70
6.7.2 Software *openBCI* 71
6.7.3 Software *MATLab* 74
6.8 Conclusões do Capítulo 74

7 Resultados Obtidos 75
7.1 Medição do Ruído no Protótipo II 75

7.2 Obtenção de Ondas Geradas Internamente pelo ADS1299 com o Protótipo II.....75
7.3 Obtenção de Sinais com o Protótipo II utilizando um Gerador de Sinais.....76
7.4 Obtenção e Análise de Sinais Fisiológicos com o Protótipo II 76
7.4.1 Resultados e Análise da Aquisição de Sinais de ECG.....77
7.4.2 Resultados e Análise da Aquisição de Sinais de EMG.....79
7.4.3 Resultados e Análise da Aquisição de Sinais EEG.....82
7.5 Conclusões do Capítulo 92

8 Conclusões e Trabalhos Futuros 93
8.1 Conclusões 93
8.2 Trabalhos Futuros 95

Referências Bibliográficas 97

Anexos.....101
Anexo A – Configuração dos Registos Internos e dos *Jumpers* do Protótipo I e II.....101
Anexo B – Configuração dos Registos Internos e dos *Jumpers* para a Medição do Nível do Ruído do Protótipo I e II 103
Anexo C – Configuração dos Registos e dos *Jumpers* para obtenção de ondas geradas internamente pelo ADS1299.....105
Anexo D – Configuração dos Registos e dos *Jumpers* para obtenção de com o protótipo I e II através de um gerador de sinais.....107
Anexo E – Código para o funcionamento do módulo *Bluetooth* no Protótipo II.....109
Anexo F – Modificação do módulo do Arduino UNO.....113
Anexo G – Configuração de Entradas Unipolares no Protótipo II 115
Anexo H – Configuração de Entradas Diferenciais no Protótipo II.....117
Anexo I – Código do Firmware do Arduíno UNO 119
Pasta Principal (*Main*): Ficheiro "StreamRawData.ino" 119
Pasta ADS1299: Ficheiro "ADS1299Manager.cpp" 130
Pasta ADS1299: Ficheiro "ADS1299Manager.h" 140
Pasta ADS1299: Ficheiro "ADS1299.cpp" 143
Pasta ADS1299: Ficheiro "ADS1299.h" 150
Pasta ADS1299: Ficheiro "Definitions.h" 151
Anexo J – Dados Demográficos dos Utilizadores do Protótipo II e dos dispositivos *Enobio NE* e *g.MOBILab+*.....153

Índice de Figuras

Figura 2.1 – Anatomia do cérebro humano. 6

Figura 2.2 – Divisão do córtex cerebral 7

Figura 2.3 – Primeiro registo da atividade elétrica produzida pelo cérebro em 1929.. 7

Figura 2.4 – Sistema Internacional 10-20: (a) Projeção lateral esquerda; e (b) Projeção superior 8

Figura 2.5 – Sistema EEG, onde é possível observar-se a touca que segura os elétrodos aplicados em posições 10-20 e os fios que os ligam à instrumentação de aquisição e de processamento dos sinais neuronais. 9

Figura 2.6 – Tipos de montagens do EEG do sistema 10-20: (a) Montagem Unipolar; e (b) Montagem Bipolar .. 10

Figura 2.7 – Exemplo da atividade do ritmo Delta. 11

Figura 2.8 – Exemplo da atividade do ritmo Teta 11

Figura 2.9 – Exemplo da atividade do ritmo Alfa 11

Figura 2.10 – Exemplo da atividade do ritmo Beta . 12

Figura 2.11 – Exemplo da atividade do ritmo Gama . 12

Figura 2.12 – Sinais de EEG consoante um individuo saudável esteja no estado: (a) excitado; 13

Figura 2.13 – Anatomia do coração humano. 14

Figura 2.14 – Onda típica de um eletrocardiograma normal. 14

Figura 2.15 – Formas do sistema de derivações proposto por Einthoven: (a) Representação visual das formas de onda; (b) Representação triangular das derivações bipolares; e (c) Representação das derivações bipolares. 16

Figura 2.16 – Terminal Central de Wilson (CT): (a) Equivalente elétrico; e (b) imagem da localização no espaço (CT'). 16

Figura 2.17 – Sistema de derivação proposto por Goldberger: (a) Exemplo da obtenção da derivação aVR pela desconexão do elétrodo do braço direito; (b) Representação das derivações unipolares aumentadas; e (c) Ilustração para obtenção das restantes derivações aVL e aVF. 17

Figura 2.18 – Sistema de derivação unipolar precordiais do plano horizontal de Wilson. 17

Figura 3.1 – Sistema de blocos funcionais de um sistema de aquisição de instrumentação médica. 21

Figura 3.2 – Dispositivo *g.tec MOBIlab+*. 24

Figura 3.3 – Dispositivo *Enobio NE*. 24

Figura 3.4 – Dispositivo *PLUX BioSignals*. 25

Figura 3.5 – Dispositivo *BITalino*, apresentando os seus diversos módulos 26

Figura 4.1 – Placa de desenvolvimento do AFE ADS1299. 29

Figura 4.2 – Diagrama de Blocos do ADS1299 31

Figura 4.3 – Diagrama físico de uma ligação SPI. 32

Figura 4.4 – Esquema de ligação de dois dispositivos em cascata. 35

Figura 4.5 – Diagrama de blocos simplificado da geração da referência interna. 37

Figura 4.6 – Diagrama simplificado para a geração de uma referência externa 38

Figura 4.7 – Esquema de um Amplificador de Ganho Programável (PGA) do ADS1299 39

Figura 4.8 – Esquema com as opções do multiplexador de entrada analógica do ADS1299. 40

Figura 4.9 – Fluxograma do funcionamento do ADS1299. 42

Figura 5.1 – Módulo desenvolvido pela TI: Placa AFE ADS1299 + DSP MMB0. 43

Figura 5.2 – Sistema de blocos funcional do módulo AFE ADS1299 + DPS MBB0. .. 44

Figura 5.3 – Interface da aplicação em *LabVIEW* desenvolvida pela empresa TI. 45

Figura 5.4 – Painel *ADC register* do AFE ADS1299: (a) *Global Channel Registers*; (b) *Channel Control Registers;* e (c) *Reference*. 46

Figura 5.5 – Painel *Analysis* que permite a visualização dos traçados dos dados obtidos. 47

Figura 5.6 – Painel *Save* que permite guardar os dados obtidos. 47

Figura 5.7 – Painel *Data Acquisition* que permite escolher o modo de adquisição de dados. 48

Figura 5.8 – Resultado do ruído obtido durante 10 segundos a 500 SPS com um ganho de 24: (a) Obtido pelo fabricante; e (b) Obtido com o protótipo I. 49

Figura 5.9 – Ondas geradas internamente pelo dispositivo ADS1299: (a) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (b) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (c) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$; e (d) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$. 50

Figura 5.10 – Sinal sinusoidal de amplitude 0,001 V e frequência de 100 Hz, a 16000 SPS. 51

Figura 5.11 – Ondas sinusoidais obtidas com o protótipo I, através do gerador de sinais: (a) Amplitude de 0,001 V e frequência de 50 Hz, a 2000 SPS; (b) Amplitude de 0,01 V e frequência de 50 Hz, a 2000 SPS; 52

Figura 6.1 – Dispositivo Arduino UNO. 56

Figura 6.2 – Dispositivo *Bluetooth* HC-05. 56

Figura 6.3 – Esquema de ligação do dispositivo *Bluetooth* HC-05 com o Arduino UNO. 57

Figura 6.4 – Ligação da placa de desenvolvimento do AFE 1299 ao módulo Arduino UNO. 58

Figura 6.5 – Kit de 8 baterias. 59

Figura 6.6 – PCB Desenhada: (a) PCB vista do Topo ; e (b) PCB vista de baixo. 59

Figura 6.7 – PCB construída: (a) Colocação do módulo Arduino UNO na PCB *Bottom*; (b) Colocação do AFE ADS1299 na PCB *Top;* e (c) PCB vista de um dos lados. 60

Figura 6.8 – Protótipo II: (a) Caixa de plástico para colocação do protótipo II; e (b) Protótipo II e o *kit* de pilhas inserido dentro da caixa. 61

Figura 6.9 – Conectores do tipo *DIN 42 804*: (a) Conectores *DIN* Fêmeas; e (b) Terminação dos elétrodos com conectores *DIN* Machos. 61

Figura 6.10 – Cabo PIN *Header* fêmea-fêmea, com o conector *DIN* 42 802 soldado numa das duas extremidades. 62

Figura 6.11 – Caixa de plástico do protótipo II: (a) Conectores *DIN* 42 802, interruptor e led colocados no exterior da caixa; e (b) Conexão dos conectores *DIN*42 802 ao ADS1299. 62

Figura 6.12 – Caixa do Protótipo II. 63

Figura 6.13 – Visual final do Protótipo II: (a) visto de cima; (b) visto de lado; (c) parte de frente; 64

Figura 6.14 – Janela inicial do software Arduino IDE. 65

Figura 6.15 – Janela *Serial Monitor* do software Arduino IDE. 71

Figura 6.16 – Interface do software desenvolvido pelo grupo *OpenSource openBCI.* 71

Figura 6.17 – Menu que permite ligar/desligar os canais do ADS1299. 72

Figura 6.18 – Visualização do valor da impedância de cada canal: (a) Menu que permite ligar/desligar a visualização do valor da impedância de cada canal; e (b) Visualização do valor da impedância de cada canal. 72

Figura 6.19 – Menu que permite escolher o sistema de filtragem, escala no domínio dos tempos e das frequências. 72

Figura 6.20 – Apresentação gráfica das medidas fisiológicas no domínio dos tempos. 73

Figura 6.21 – Apresentação gráfica das medidas fisiológicas no domínio das frequências. 73

Figura 6.22 – Posicionamento dos elétrodos indicando a sua intensidade de sinal. 73

Figura 6.23 – Formato do ficheiro de texto onde são guardados os valores das medidas fisiológicas adquiridas..... 74

Figura 7.1 – Ruído medido com o Protótipo II..... 75

Figura 7.2 – Ondas quadradas geradas internamente no protótipo II pelo dispositivo ADS1299: (a) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (b) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (c) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$; e (d) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$. 76

Figura 7.3 – Elétrodo ECG Tripolar: (a) Elétrodo dos dispositivos *PLUX* e *BITalino;* e (b) Autocolante que permite uma melhor aderência dos elétrodos à pele. 77

Figura 7.4 – Posicionamento do elétrodo tripolar na zona ventricular V2 do utilizador. 77

Figura 7.5 – Elétrodo opcional colocado na perna direita (tornozelo) do utilizador, com a função de reduzir o ruído do sinal ECG. 78

Figura 7.6 – Sinais de ECG obtidos com: (a) Protótipo II; (b) Dispositivo *PLUX;* e 79

Figura 7.7 – Elétrodos EMG superficiais: (a) Dispositivo *PLUX* e (b) Dispositivo *BITalino.* 80

Figura 7.8 – Autocolantes dos elétrodos de EMG: (a) Face que permite a ligação do elétrodo e (b) Face que permite aderir à pele do utilizador. 80

Figura 7.9 – Colocação dos elétrodos de EEG no músculo *bíceps* do utilizador, para a obtenção de sinais de EMG..... 81

Figura 7.10 – Sinais de EMG: (a) Obtido com o Protótipo II através do software *openBCI* e (b) Obtido com o dispositivo *PLUX* e pós-processamento com a ferramenta *MATLab*. 81

Figura 7.11 – Posicionamento dos elétrodos para a visualização de artefactos dos sinais de EEG. 83

Figura 7.12 – Clip que permite a colocação dos elétrodos de referência e de *ground* no lóbulo da orelha do utilizador. 83

Figura 7.13 – Capacete de adquisição de sinais de EEG: (a) Visto da parte de frente; (b) Visto da parte de trás; e (c) Visto de um dos lados. 83

Figura 7.14 – Sinal EEG obtido com o protótipo II, no estado de *baseline.* 84

Figura 7.15 – Sinal EEG obtido com o protótipo II, no estado de *blink.* 85

Figura 7.16 – Sinal EEG obtido com o protótipo II, no estado de *roll eyes.* 86

Figura 7.17 – Sinal EEG obtido com o protótipo II, no estado de *open/close mouth.* ... 86

Figura 7.18 – Sinal EEG que obteve-se com o protótipo II: (a) No estado de *move hands;* e (b) No estado de *move feet*. 87

Figura 7.19 – Interface mostrada ao utilizador na etapa de treino: (a) Ecrã inicial; e (b) Ecrã mostrando uma seta para a direita. 89

Figura 7.20 – Posicionamento dos elétrodos para a utilização do método *Motor Imagery*. 89

Figura 7.21 – Capacete de EEG colocado na cabeça do utilizador: (a) Visto de frente; (b) Visto de cima; e (c) Visto do lado direito. 90

Figura 7.22 – Interface mostrada ao utilizador na etapa online: (a) Seta e barra azul coincidindo, indicando o acerto do utilizador; e (b) Seta e barra azul com sentidos opostos, indicando o erro do utilizador. 91

Índice de Tabelas

Tabela 2.1 – Caraterísticas de amplitude e frequência das variáveis fisiológicas em estudo. 5

Tabela 2.2 – Categorização dos ritmos de EEG no domínio das frequências, e estados de ocorrência não patológica. 10

Tabela 2.3 – As 12 derivações-padrão de um ECG 15

Tabela 3.1 – Comparação de alguns dispositivos de medição fisiológica existentes no mercado. 27

Tabela 4.1 – Mapa de Configuração de registos CONFIG1. 35

Tabela 4.2 – Relógio do sistema ADS1299. 35

Tabela 4.3 – *Bits*[DR] que podem ser ajustados para se obter a taxa de amostragem desejada. 36

Tabela 4.4 – Mapa de configuração de registos CONFIG2. 36

Tabela 4.5 – Mapa de configuração de registos CONFIG3. 37

Tabela 4.6 – Mapa de configuração de registos CHnSET. 38

Tabela 4.7 – Configuração dos ganhos disponíveis no ADS1299. 39

Tabela 4.8 – Opções das entradas analógicas disponíveis no ADS1299. 40

Tabela 4.9 – Mapa de configuração de registos MISC1. 41

Tabela 6.1 – Correspondência dos pinos de comunicação SPI do ADS1299 e do Arduino UNO. 58

Tabela 6.2 – Descrição e preço dos dispositivos e componentes do Protótipo II. 65

Tabela 6.3 – Comandos SPI que podem ser enviados pelo utilizador do Protótipo II. 68

Tabela 6.4 – Registos alterados no software. 69

Tabela 7.1 – Resultados obtidos na etapa de Treino com o Protótipo II. 90

Tabela 7.2 – Resultados obtidos na etapa Online, com o Protótipo II. 91

Acrónimos

ADC	*Analog to Digital Converter*
AFE	*Analog Front End*
ALS	*Amyotrophic Lateral Sclerosis*
BCI	*Brain Computer Interface*
CMRR	*Common-Mode Rejection Ratio*
CS	*Chip Select*
DSP	*Digital Signal Processor*
ECG	*Electrocardiogram*
ECoG	*Electrocorticography*
EDA	*Eletrodermal*
EEG	*Electroencephalography*
EMG	*Electromyography*
ERP	*Event Related Potencial*
FES	*Funtional Electric Stimulation*
IFSECN	*International Federation of Societies for Electroencephalography and Clinical Neurophysiology*
LA	*Left Arm*
LL	*Left Leg*
LSB	*Least Significal Bit*
M-ITI	*Madeira Interactive Technologies Institute*
MCU	*Multipoint Control Unit*
MISO	*Master Input Slave Output*
MOSI	*Master Output Slave Input*
MSB	*Most Significal Bit*
PGA	*Programmable Gain Amplifier*
RA	*Rigth Arm*
RL	*Rigth Leg*
SPI	*Serial Peripheral Interface*
SPS	*Sample Per Second*
TI	*Texas Instruments*

1 Introdução

Este primeiro capítulo descreve o contexto e a motivação que levaram à escolha do tema desta dissertação, dos seus objetivos e à sua estruturação.

1.1 Contexto e Motivação

Segundo a Organização Mundial da Saúde (OMS), mais de um bilião de pessoas em todo o mundo são afetadas por distúrbios neurológicos, que incluem a epilepsia, a enxaqueca, o autismo e desordens neuro-degenerativas (*Alzheimer*, *Parkinson* e a esclerose múltipla), independentemente do seu pais origem, idade e sexo. À medida que a população mundial e a esperança média de vida aumentam em todo o mundo, a tendência dos distúrbios neurológicos nas pessoas também acompanham esse crescimento.

Ao longo dos últimos anos tem sido desenvolvido a nível mundial trabalho relacionado com a produção de sistemas de alto desempenho, que permitam a obtenção de medidas fisiológicas, também chamadas de sinais vitais, em seres humanos, nomeadamente, o electroencefalograma (EEG), o electrocardiograma (ECG) e o electromiograma (EMG), como sistemas para aplicações em diversas áreas como a medicina, a indústria e a investigação. A maior parte do trabalho/investigação realizado na indústria é direcionado principalmente para a comunidade de jogadores, destacando-se os trabalhos realizados pela *NeuroSky*, *Emotiv Headset*, *BioSignals* e o *OpenEEG Project*.

Atualmente existem no mercado diversos dispositivos que permitem a gravação de sinais com alguma fidelidade, no entanto o custo dos mesmos é muito elevado, e só dispositivos de valor acima dos 5000 € é que permitem a aquisição dos sinais fisiológicos mais importantes (EEG, ECG e EMG). Este projeto visava encontrar uma solução para esse problema, através do desenvolvimento de um sistema capaz de adquirir estas medidas fisiológicas, com a mesma qualidade e fiabilidade que os produtos comerciais, mas a um preço muito mais reduzido, na ordem dos 300 €. O dispositivo deverá ser capaz de adquirir os diversos sinais, ser de fácil manuseamento, de utilização intuitiva e servirá assim para futuros projetos a desenvolver na instituição Universidade da Madeira (UMa) e *Madeira Interactive Technologies Institute* (M-ITI).

1.2 Objetivos

A medição de padrões cerebrais é uma área de enorme interesse não só na área da medicina, mas também em áreas associadas tais como a reabilitação. Neste projeto pretendeu-se construir um sistema de baixo custo, com baixo ruído e alto desempenho, que permitisse a medição desses padrões, bem como de sinais cuja aquisição é geralmente mais fácil como o ECG e o EMG. O sistema desenvolvido deveria facilitar a interação cérebro-computador (*Brain Computer Interface* – BCI), permitindo futuramente desenvolver outros projetos, como a criação de paradigmas de treino mental do tipo *neurofeedback* para pessoas com danos cerebrais.

Os principais objetivos deste trabalho são enumerados de seguida:

Estudo das medidas fisiológicas mais importantes nos seres humanos, nomeadamente, EEG, ECG, e EMG, dando uma maior enfâse aos sinais EEG. Pesquisa de dispositivos que permitam esse tipo de medições, examinando as suas principais caraterísticas.

Escolha de um sistema que possibilite o desenvolvimento de um protótipo que realize as medições fisiológicas referidas no primeiro ponto.

Construção do protótipo, criação e/ou adaptação do software de leitura e visualização de dados fisiológicos em tempo real. Realização de testes para verificação do correto funcionamento.

Comparação com outros dispositivos existentes na instituição, nomeadamente os produtos comerciais *g.tec MOBIlab+*, *Enobio NE*, *PLUX BioSignals* e *BITalino BioSignals*, para avaliar as limitações do sistema desenvolvido.

1.3 Estrutura da Tese

Nesta secção é apresentada a estrutura desta dissertação, a qual se encontra dividida em 8 capítulos, organizados em várias secções, para uma melhor percepção por parte do leitor. A estruturação encontra-se organizada da forma que se segue:

Capítulo I – Introdução: Neste capítulo é apresentado o contexto e a motivação que levaram à escolha do tema, bem como os objetivos a cumprir no decorrer desta dissertação.

Capítulo II – Medidas Fisiológicas: No segundo capítulo desta dissertação são introduzidos os conceitos essenciais para a compreensão da aquisição de medidas fisiológicas num ser humano.

Capítulo III – Instrumentação Médica e Dispositivos de Medição Fisiológica: Neste capítulo são apresentados conceitos de elevada importância e que devem ser considerados quando se está a trabalhar na área da instrumentação médica, nomeadamente na obtenção de sinais fisiológicos, e são ainda apresentados vários dispositivos que atualmente se encontram no mercado e no Laboratório da UMa/M-ITI que permitem este tipo de medições fisiológicas.

Capítulo IV – Dispositivo ADS1299: Serão abordadas neste capítulo as principais características do dispositivo escolhido para a criação do protótipo deste projeto.

Capítulo V – Protótipo I: No quinto capítulo é apresentado o primeiro protótipo estudado, nomeadamente, o seu módulo de adquisição de dados e a interface do sistema com o computador.

Capítulo VI – Protótipo II: O segundo protótipo, que corresponde à versão final do sistema desta dissertação, é apresentado neste capítulo, sendo indicados todos os passos desde a sua criação até à obtenção de dados fisiológicos.

Capítulo VII – Resultados Obtidos com o Protótipo II: Neste capítulo são descritos os testes e são apresentados os resultados, seguidos da respetiva análise. Ainda são expostas

algumas comparações entre o protótipo desenvolvido e os dispositivos apresentados no final do capítulo 3.

Capítulo VIII – Conclusões e Trabalhos Futuros: Neste capítulo são apresentadas as conclusões finais, tendo em conta a contribuição do dispositivo desenvolvido. São também referidos trabalhos a realizar futuramente com vista à otimização do projeto.

2 Medidas Fisiológicas

Neste capítulo é introduzido o conceito de variável fisiológica e os diversos tipos de medidas fisiológicas mais comuns (EGG, ECG e EMG), apresentando-se as suas diversas características, formas de medição, entre outros fatores de elevada importância para a compreensão desta dissertação.

2.1 Variáveis Fisiológicas

Neste contexto entendem-se por variáveis fisiológicas aquelas que podem ser registadas continuamente ou através do registo da diferença de potencial elétrico entre dois pontos do corpo humano [1]. A maioria das variáveis fisiológicas possui amplitudes bastante reduzidas, inferiores a 1 V e surgem ao longo dos tecidos corporais, cuja constituição são as células. Estas exibem uma diferença de potencial entre o seu interior e o meio externo quando se encontram em repouso, sendo que quando é excitado um ponto da célula o resultado é a criação de dipolos e a propagação do campo elétrico através dos tecidos, alcançando a superfície da pele. De entre todos os tipos de células, as mais excitáveis são as células musculares e as nervosas [2].

2.2 Tipos de Medidas Fisiológicas

A Tabela 2.1 resume as características de amplitude e espectro de frequência das variáveis fisiológicas que serão abordadas nesta dissertação. Nas seguintes secções são explicadas algumas características destas variáveis, assim como os seus métodos de medição convencionais.

Tabela 2.1 – Caraterísticas de amplitude e frequência das variáveis fisiológicas em estudo [1].

Variável Fisiológica	Amplitude (μV)	Frequência (Hz)
EEG	1 – 100	0,3 – 70
ECG	500 – 4000	0,3 – 70
EMG	60000	> 42

2.3 Eletroencefalograma (EEG)

Para uma melhor percepção do que será apresentado nesta secção, inicialmente será exposta uma breve explicação da anatomia do cérebro humano, seguindo-se depois a descrição de um eletroencefalograma.

2.3.1 Anatomia do Cérebro Humano

O cérebro humano é o centro do sistema nervoso humano. É o cérebro que monitoriza, deteta, e regula as operações do corpo [3]. Na Figura 2.1 pode-se visualizar a anatomia do cérebro humano.

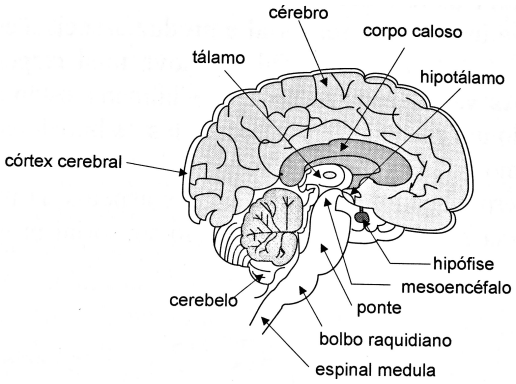


Figura 2.1 – Anatomia do cérebro humano [3].

O cérebro é composto por quatro divisões principais [3,4,5]:

Tronco Encefálico (*Brain Stem*): É constituído pela medula oblongata, cérebro posterior e mesencéfalo. É responsável por regular o batimento cardíaco, a circulação do sangue, a respiração e outras funções automáticas que não precisam do processamento ativo do cérebro.

Cerebelo (*Cerebellum*): É chamado o pequeno cérebro. É uma parte importante do cérebro que desempenha um papel na aprendizagem do movimento motor. Ajuda a manter o equilíbrio do corpo e a postura. Os impactos na coordenação motora são significativos quando esta parte é danificada.

Sistema Límbico (*Limbic System*): O sistema límbico é chamado o cérebro emocional. Está envolvido na emoção, aprendizagem, motivação e memória. Contém o tálamo, hipotálamo, amígdala e o hipocampo.

Córtex Cerebral (*Cerebral Cortex*): O córtex cerebral abrange 80% do cérebro humano. É a região que permite o pensamento abstrato e simbólico. Está dividido em hemisférios e permite a comunicação entre diferentes partes do cérebro. A parte esquerda do cérebro controla o lado direito do corpo e vice-versa. O hemisfério esquerdo é também responsável pela linguagem, lógica e comportamento motor, enquanto o lado direito é usado no reconhecimento de rostos, lugares, som e música. O córtex cerebral é dividido em quatro lóbulos diferentes (Figura 2.2), que são os seguintes [4,5]:

Lóbulo Frontal: É o maior dos quatro lóbulos e ajuda no planeamento do movimento, pensamento abstrato e resolução de problemas.

Lóbulo Temporal: É responsável pela audição, linguagem e memória.

Lóbulo Occipital: É responsável pela visão.

Lóbulo Parietal: É responsável pela percepção de estímulos, como o toque, pressão, temperatura e dor.

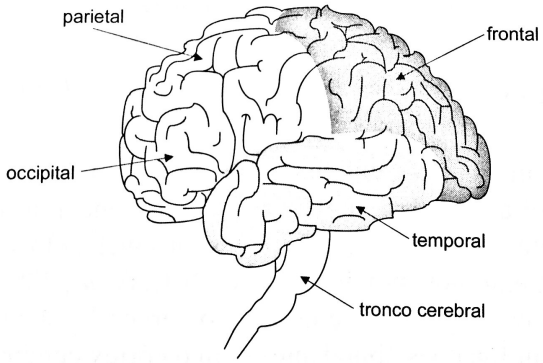


Figura 2.2 – Divisão do córtex cerebral [6].

2.3.2 Descrição de um EEG

A palavra eletroencefalograma deriva da composição das palavras gregas *enkephalo* (cérebro) e *graphein* (escrever) [7]. Um eletroencefalograma (EEG) é um registo do sinal elétrico gerado pela ação integrada dos neurónios. O que é medido pelo EEG são as diferenças de potencial elétrico entre diferentes regiões do cérebro [7,8].

O primeiro registo da atividade elétrica produzida pelo cérebro (Figura 2.3) foi alcançado em 1929, pelo psiquiatra alemão Hans Berger. Este anunciou ao mundo científico que tinha sido capaz de detetar e registar fracas correntes elétricas produzidas pelo cérebro, utilizando um método não invasivo. Demostrou ainda que a atividade variava de acordo com o estado funcional do cérebro. Este passo dado por Berger contribuiu para a criação de um novo ramo nas ciências médicas, a neurofisiologia clínica. [7,8].

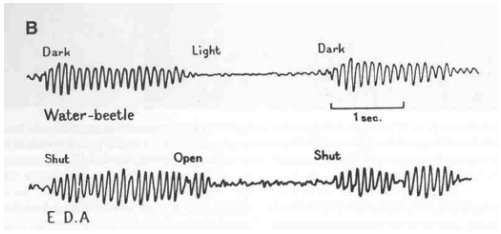


Figura 2.3 – Primeiro registo da atividade elétrica produzida pelo cérebro em 1929 [7].

Os resultados obtidos naquela época são em tudo semelhantes aos obtidos nos dias de hoje, contudo Berger não disponha das tecnologias e ferramentas atuais, utilizadas para visualizar e registar o EEG [7,8,9]. Os sinais de EEG podem ser adquiridos através de três tipos de medidas:

Atividade espontânea: A atividade espontânea, ou eletrocorticograma (ECoG), corresponde ao registo do comportamento natural da atividade elétrica do cérebro.

Potenciais externos: Os potenciais externos, também conhecidos por *Event-Related-Potential* (ERP), são a reposta elétrica do cérebro a determinados estímulos induzidos (que podem ser elétricos, auditivos, visuais, entre outros). Neste tipo de registo os resultados são bastantes difíceis de identificar, e confundem-se com o ruído predominante nas medidas,

obrigando ao uso de sequências de estímulos, e ao processamento de sinal para melhorar a relação sinal-ruído, tornando a resposta aos estímulos mais perceptível.

Atividade elétrica de um neurónio: A medida da atividade elétrica de um neurónio apenas é possível detetar com microeléctrodos [9].

A amplitude de um sinal EEG gravado com elétrodos sobre o couro cabeludo de um indivíduo normal e acordado situa-se na gama dos 10–150 μV. No caso de um paciente epilético as amplitudes do EEG podem aumentar cerca de uma ordem de grandeza. Para elétrodos colocados sobre o córtex, as amplitudes de sinal são de intensidade superior.

Tanto o limite inferior como o superior da banda de frequências do sinal de EEG com interesse clínico e científico são subjetivos. Por exemplo, as componentes de muito baixa frequência (inferiores a 1 Hz) são frequentemente utilizadas para situações de estados de coma profundo e em estados terminais. Geralmente a resposta em frequência de um sistema de adquisição de EEG concentra-se na banda considerada mais relevante do ponto de vista clínico, psicofisiológico e científico (0,05–150 Hz) [8,9].

2.3.3 Sistema 10-20 e Montagem de Elétrodos

O processo de aquisição e registo de sinais de EEG necessita de elétrodos capazes de medir as diferenças de potencial entre os pontos do couro cabeludo com relevância para o estudo em causa. Um organismo internacional denominado *International Federation of Societies for Electroencephalography and Clinical Neurophysiology* (IFSECN) foi estabelecido com o objetivo de definir e apresentar recomendações relacionadas com a aquisição de sinais de EEG. Desse trabalho resultou a proposta de um sistema padrão de colocação de elétrodos para permitir a replicação de estudos ao longo do tempo e em ambientes clínicos e/ou laboratoriais diferentes.

Esse sistema está ilustrado na Figura 2.4 e é atualmente conhecido como Sistema Internacional 10-20, pois baseia-se em 10 a 20 % das distâncias entre alguns marcos ósseos como distâncias de referências entre elétrodos, para determinar as suas posições relativas [10]. Na Figura 2.4, pode observar-se que a colocação dos elétrodos deve ser feita ao longo dos meridianos de forma a que a soma das distâncias entre os elétrodos totalize 100 % do comprimento total do meridiano.

(a) (b)

Figura 2.4 – Sistema Internacional 10-20: (a) Projeção lateral esquerda; e (b) Projeção superior [11].

O sistema de numeração do sistema 10-20 determina que os elétrodos com número ímpar se localizam no hemisfério esquerdo, enquanto os números pares ficam reservados para o hemisfério direito. Além disso, utiliza-se a letra "Z" como índice de referência aos elétrodos posicionados nos hemisférios. Por sua vez as áreas anatómicas de posicionamento dos elétrodos identificam-se por um conjunto de 5 letras. Assim, o conjunto formado por (F, T, C, P, O), designa, respetivamente, os lóbulos Frontal, Temporal, Central, Parietal e Occipital [11,12]. A Figura 2.5, exemplifica uma realização prática de um sistema EEG, na qual é possível observar-se a touca que segura os elétrodos aplicados em posições definidas pelo sistema 10-20 da IFSECN e os respetivos fios que os ligam à instrumentação de aquisição e de processamento.

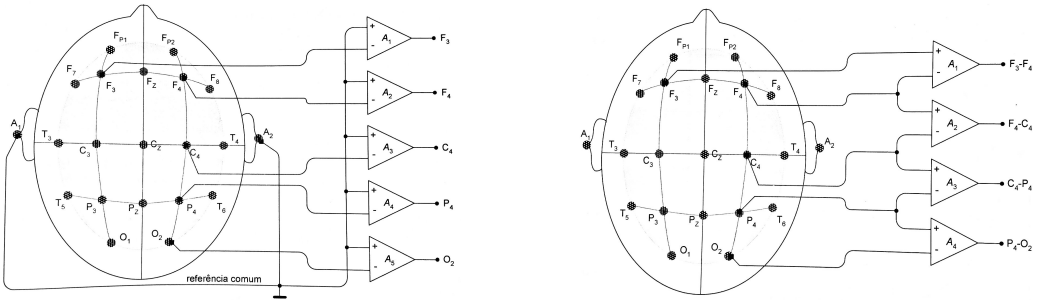
Figura 2.5 – Sistema EEG, onde é possível observar-se a touca que segura os elétrodos aplicados em posições 10-20 e os fios que os ligam à instrumentação de aquisição e de processamento dos sinais neuronais.

2.3.4 Processamento de Sinal do EEG

Antes de se efetuar a aquisição do sinal do EEG, é necessário definir a combinação de elétrodos (ou seja, a montagem). A montagem utilizada tem por objetivo uma mais fácil interpretação das caraterísticas do sinal EEG a analisar. Conforme ilustrado na Figura 2.6, uma montagem pode-se classificar como sendo unipolar (isto é, referencial), ou do tipo bipolar. Numa montagem unipolar (Figura 2.6-a), as diferenças de potencial são medidas entre cada elétrodo e um elétrodo de referência comum (normalmente numa posição com o mínimo de atividade possível, ou seja, num dos dois pontos pré-auriculares, localizados no lóbulo de cada orelha). Caso se opte por uma montagem bipolar, então as diferenças de potenciais medem-se entre pares de elétrodos.

As montagens bipolares (Figura 2.6-b) assumem especial interesse nas situações de medição de caraterísticas do EEG que se manifestam de forma distinta em cada hemisfério cerebral. Nesse caso, a implementação da montagem é de modo a que cada canal meça as diferenças de potencial entre elétrodos de hemisférios diferentes. Todavia uma montagem bipolar apresenta desvantagens na aquisição de potenciais em situações onde haja uma grande distribuição espacial. Nesse caso a montagem unipolar constitui uma alternativa melhor, embora deva haver o cuidado de ter em conta que, independentemente do elétrodo usado

como referência, os potenciais registados numa dada posição vão ser sempre influenciados pelas outras posições simultaneamente adquiridas [12].



(a) (b)

Figura 2.6 – Tipos de montagens do EEG do sistema 10-20: (a) Montagem Unipolar; e (b) Montagem Bipolar [12].

2.3.5 Ritmos do EEG

O sinal de EEG registado à superfície do couro cabeludo carateriza-se por ter uma amplitude média de 1 a 2 μV. A interpretação do EEG é feita a partir dos ritmos (Alfa, Beta, Delta, Gama e Teta), cada um com uma gama de frequências diferente, que pode variar dos 0,3 aos 70 Hz [13]. A Tabela 2.2 resume as caraterísticas das bandas de frequência mais utilizadas na adquisição de sinais EEG.

Tabela 2.2 – Categorização dos ritmos de EEG no domínio das frequências, e estados de ocorrência não patológica [13].

Ritmo	Banda (Hz)	Estados de Ocorrência
Delta	0,3 – 3	Em adultos durante o sono profundo ou em crianças bastante novas
Teta	3 – 7	Em todas as fases de sono e em crianças
Alfa	7 – 13	Em todas as faixas etárias, embora sejam mais comuns em adultos; mais evidente com os olhos fechados, desaparecendo normalmente com a atenção
Beta	13 – 30	Em qualquer idade no estado acordado
Gama	> 30	Na execução de movimentos

O significado de um determinado ritmo cerebral obtém-se combinando um conjunto de fatores relevantes dos sinais de EEG, tais como, a frequência, a forma de onda, a idade do indivíduo, o estado de alerta e o local de gravação. Todavia, a frequência é a principal característica na distinção dos ritmos normais e anormais num EEG. Por exemplo, as frequências superiores a 7 Hz são consideradas normais no EEG de um adulto acordado, enquanto as frequências inferiores a 7 Hz são frequentemente associadas a situações anormais em adultos acordados, embora possam ser detetadas em crianças ou adultos durante o sono [13,14]. As componentes de baixa frequência não são críticas pois relacionam-se com artefactos com origem nos movimentos oculares, respiração ou batimento cardíaco.

O ritmo Delta (ou ondas Delta) possui uma frequência inferior a 3 Hz e regista-se normalmente em adultos durante o sono profundo, em bebés e crianças. O traço da Figura 2.7, refere-se a segmentos de sinais de EEG gravados num indivíduo em sono profundo. Note-se ainda que as ondas Delta representam um mistério para a ciência, pois é o ritmo cerebral que ocorre no estado de coma, em catalepsia e em sono muito profundo [14].

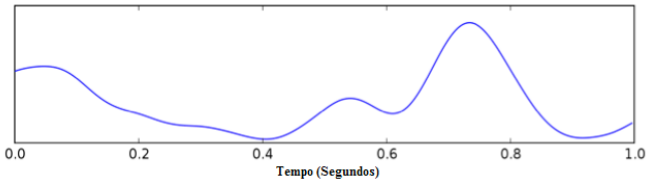


Figura 2.7 – Exemplo da atividade do ritmo Delta [15].

O ritmo Teta, Figura 2.8, está compreendido na gama de frequências dos 3 aos 7 Hz, e é associado à sonolência, normalmente de crianças e adolescentes. Este ritmo pode ser encontrado em estados de transe, hipnose, sonhos diurnos, sonhos lúcidos e sono leve, no estado pré-consciente antes de acordar e durante o adormecer [14].

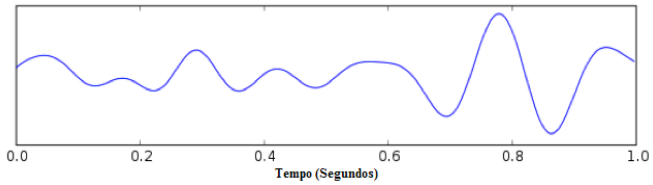


Figura 2.8 – Exemplo da atividade do ritmo Teta [15].

Relativamente ao ritmo Alfa, as suas frequências situam-se entre os 7 e os 13 Hz, sendo geralmente registadas em sujeitos de todas as faixas etárias, embora o seu registo seja mais comum quando efetuado em sujeitos adultos. Embora os ritmos das ondas Alfa ocorram em ambos os lados da cabeça, frequentemente é no lado contrário à destreza do indivíduo que se observam as amplitudes mais elevadas. Este efeito é mais notório em indivíduos destros. Além disso é importante referir que as ondas Alfa estão presentes principalmente nas zonas posteriores da cabeça, sendo a sua amplitude especialmente elevada quando os indivíduos têm os olhos fechados ou estão num estado de relaxamento. O traço da Figura 2.9, refere-se a segmentos de sinais de EEG gravados num indivíduo acordado em nível Alfa. As ondas Alfa estão portanto relacionadas com o nível de tranquilidade. Todavia, a atividade Alfa desaparece quando o individuo presta atenção a uma determinada tarefa mental (isto é, uma tarefa que envolve um exercício mental e não produz resposta comportamental, nomeadamente a execução de aritmética mental, imaginação de movimentos, imaginação de objetos em rotação, entre outras) [14].

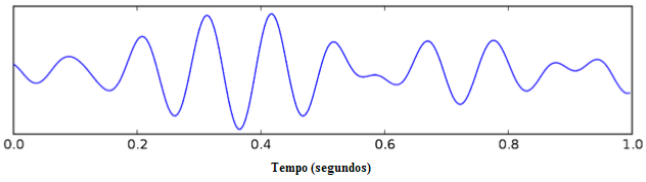


Figura 2.9 – Exemplo da atividade do ritmo Alfa [15].

O ritmo Beta (ou ondas Beta) é observado em todas as faixas etárias, e apresenta amplitudes baixas. As ondas Beta podem apresentar múltiplas frequências, entre os 13 e os 30 Hz, em resposta à contínua atividade cerebral, ansiedade ou concentração, embora sejam geralmente associadas a um estado normal de consciência em indivíduos acordados (ou seja, quando os cinco sentidos estão em atividade). O traço da Figura 2.10 refere-se a segmentos de sinais de EEG gravados num indivíduo em nível Beta [14].

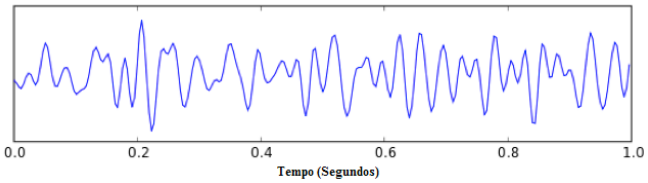


Figura 2.10 – Exemplo da atividade do ritmo Beta [15].

Finalmente, as ondas Gama correspondem às atividades rítmicas com frequências superiores a 30 Hz (Figura 2.11). Um estudo das ondas Gama demostrou uma atividade rítmica de 40 Hz sobre a região central esquerda do cérebro, em reação ao começo de movimentos voluntários do dedo direito, durante o período de preparação do movimento e uma elevada sincronização da atividade de 40 Hz, que coincidiu com o início do movimento. Em suma, estes estudos sugerem que as ondas Gama possam estar relacionadas com o controlo fino de movimentos voluntários [14].

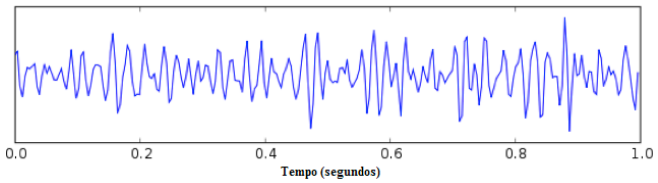


Figura 2.11 – Exemplo da atividade do ritmo Gama [15].

2.3.6 Alterações do EEG

Os sinais de EEG permitem ainda aferir acerca dos estados de consciência de indivíduos saudáveis, ou seja, analisando conjuntamente a frequência e a amplitude dos sinais consegue-se saber em que estado se encontram. A Figura 2.12, ilustra as alterações verificadas na frequência e amplitude dos sinais EEG, sendo possível observar claramente uma diminuição progressiva da frequência das ondas cerebrais à medida que o indivíduo experimenta sucessivamente os estados de excitação, relaxamento, sonolência, sono e sono profundo.

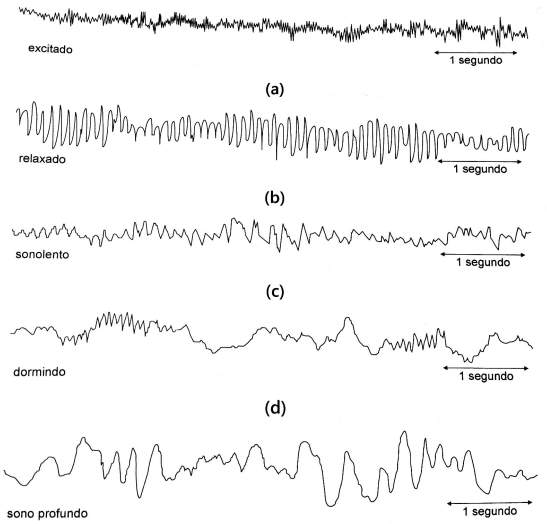


Figura 2.12 – Sinais de EEG consoante um individuo saudável esteja no estado: (a) excitado; (b) relaxado; (c) sonolento; (d) dormindo; e (e) mergulhado em sono profundo [16].

Durante os últimos anos tem-se intensificado o interesse pelos ritmos ultrarápidos, limitados inferiormente a 80 Hz e superiormente a 1000 Hz. O aparecimento da gravação digital de EEG abriu novas portas a esta tendência. Essa escala de frequências é especialmente importante para o estudo da epilepsia e para a compreensão da percepção cortical, e da atividade motora com relevância nos processos neuro-cognitivos [16].

2.4 Eletrocardiograma (ECG)

Tal como foi feito para o EEG, nesta secção será inicialmente apresentada uma breve explicação da anatomia do coração humano, seguindo-se a descrição do eletrocardiograma.

2.4.1 Anatomia do Coração Humano

O coração é constituído por dois tipos principais de células: musculares e especializadas do sistema de condução. As primeiras têm função contráctil e desempenham o trabalho mecânico da contração: as quatro câmaras cardíacas (aurícula direita e esquerda e ventrículo direito e esquerdo) que podem ser visualizadas na Figura 2.13, são formadas pelas células musculares que após a *diástole* (período de repouso durante o qual as câmaras acumulam o sangue) são estimuladas durante a *sístole* para encurtar, propulsionando o sangue para o sistema cardiovascular. O segundo tipo de células não é responsável pelo trabalho mecânico da contração, mas coordenam a atividade, iniciando o ritmo cardíaco e propagando o impulso elétrico através do estímulo da contração das células miocardias [7,8,9,17].

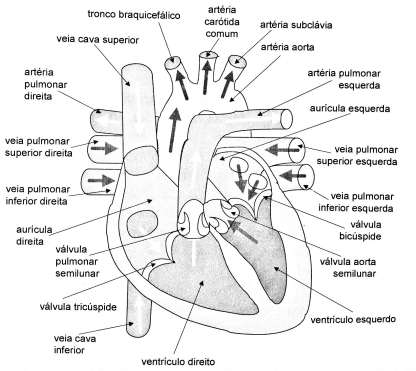


Figura 2.13 – Anatomia do coração humano [17].

A atividade elétrica do coração deve-se ao facto de as células cardíacas serem carregadas ou polarizadas no estado de repouso, mas quando eletricamente estimuladas, "despolarizam-se" e contraem-se. Assim, a contração do miocárdio é produzida por uma onda progressiva de estimulação (despolarização) que atravessa o coração. As ondas de despolarização (o interior das células torna-se positivo) e a de repolarização (as células voltam a ser negativas) ao atravessarem o coração são captadas pelos detetores externos (cutâneos), registando-se essa atividade num ECG. Quando a onda positiva de despolarização dentro das células cardíacas se move em direção ao elétrodo positivo (pele), regista-se sobre o ECG uma deflexão positiva [17].

Como se pode observar na

Figura 2.14, o eletrocardiograma é geralmente composto por vários segmentos de sinal (chamados de complexos): P, Q, R, S (Q+R+S=complexo QRS), T e U.

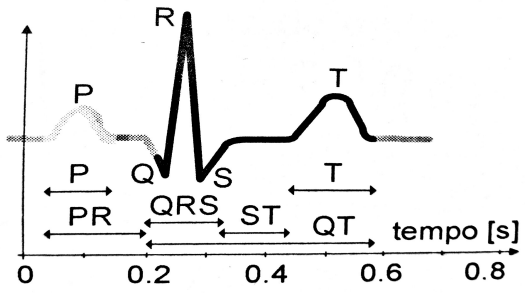


Figura 2.14 – Onda típica de um eletrocardiograma normal [17].

2.4.2 Descrição do ECG

O eletrocardiograma (ECG) é um gráfico da atividade elétrica do coração, que se propaga até à superfície corporal, acompanhando a sua atividade mecânica. Este método de medida é utilizado no diagnóstico de doenças cardiovasculares, uma vez que se trata de um método simples e não invasivo, causando pouco desconforto, sendo adequado a longos períodos de medida, e ainda apresentando custos reduzidos [7,8]. A formação do sinal ECG, durante o ciclo cardíaco, resulta dos potenciais de ação (processo de excitação e repolarização) das fibras do miocárdio, que refletem os eventos associados ao ciclo mecânico cardíaco [8].

O primeiro registo do potencial elétrico associado ao ciclo cardíaco realizado sobre a superfície corporal, foi obtido por Augustus Desiree Waller, em 1897, utilizando um eletrómetro, um instrumento de medida, cujo funcionamento baseia-se na deteção da diferença de tensão superficial entre líquidos colocados num tubo (mercúrio e ácido sulfúrico) [9].

2.4.3 Potenciais de Superfície e Aquisição de Sinais de ECG

Existe um sistema padrão que define os pontos de colocação e posicionamento dos elétrodos de um ECG. Este sistema foi implementado por Einthoven, com base no pressuposto do coração ser o centro de uma esfera homogénea representativa do torso (mais conhecido por tronco, o qual é constituído pelo tórax e pelo abdómen). A operação de posicionamento dos elétrodos do ECG depende dos potenciais que se pretendem medir (também conhecidos como derivações) e dos dispositivos de aquisição.

Os sinais de ECG podem ser adquiridos com recurso a dois tipos de ligações, nomeadamente as derivações bipolares e as unipolares. As derivações bipolares medem a diferença de potencial entre dois pontos (positivo e negativo) do corpo, enquanto que as derivações unipolares medem os potenciais absolutos (positivos) relativamente a um ponto comum (designado por referência) [18]. Em estudos específicos consideram-se doze derivações-padrões do ECG, as quais se encontram descritas na Tabela 2.3. Essas derivações são organizadas em dois planos, nomeadamente o plano frontal (as primeiras 6 derivações) e o plano horizontal (as últimas 6 derivações).

Tabela 2.3 – As 12 derivações-padrão de um ECG [18,19].

Derivação	Localização
I	Braço direito, braço esquerdo
II	Braço direito, perna esquerda
III	Braço esquerdo, perna esquerda
aVL	Amplificação do potencial detetado no braço esquerdo
aVR	Amplificação do potencial detetado no braço direito
aVF	Amplificação do potencial detetado no membro inferior esquerdo
V1	Quarto espaço intercostal, bordo esternal direito
V2	Quarto espaço intercostal, bordo esternal esquerdo
V3	Ponto intermédio entre V_2 e V_4
V4	Linha médio-clavicular esquerda no quinto espaço intercostal
V5	Mesmo plano de V_4 na linha axial anterior
V6	Mesmo plano de V_4 na linha axial média

Os elétrodos podem ocupar quatro posições quando se considera o plano frontal. Essas posições coincidem com as quatro extremidades formadas pelos membros superiores (os

braços) e pelos membros inferiores (as pernas), isto é, o braço direito (*Rigth Arm* – RA), o braço esquerdo (*Left Arm* – LA), a perna direita (*Rigth Leg* – RL) e a perna esquerda (*Left Leg* – LL). Estas quatro posições permitem obter as três derivações bipolares (I, II e III) e a três unipolares (aVR, aVL e aVF) de Einthovem [18,19,20], representadas na Figura 2.15.

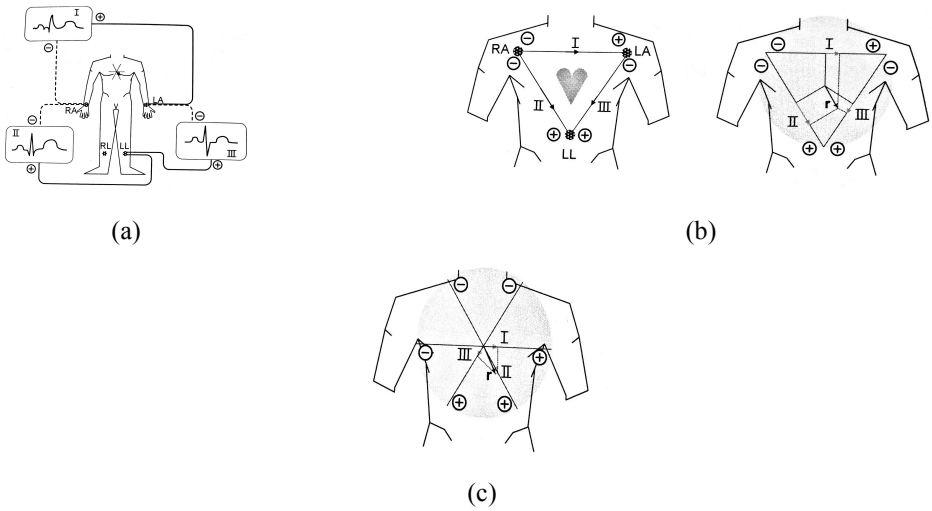


Figura 2.15 – Formas do sistema de derivações proposto por Einthoven: (a) Representação visual das formas de onda; (b) Representação triangular das derivações bipolares; e (c) Representação das derivações bipolares [20].

Observando a Figura 2.15-a, a derivação I permite adquirir o sinal correspondente à diferença de potencial entre LA e RA. Por outro lado, a derivação II permite adquirir o sinal correspondente à diferença de potencial entre LL e RA, enquanto que a derivação III permite adquirir o sinal correspondente à diferença de potencial entre LL e LA. Por outro lado observando a Figura 2.15-b, os elétrodos nas posições RA, LA e LL permitem também medir três potenciais absolutos relativamente a uma referência remota – teoricamente localizada no infinito. Esta referência remota foi obtida a partir da ligação de uma resistência de 5 kΩ entre cada um dos três elétrodos e um ponto comum (Figura 2.16), que atualmente é conhecido por Terminal Central de Wilson (CT) [21,22,23].

Figura 2.16 – Terminal Central de Wilson (CT): (a) Equivalente elétrico; e (b) imagem da localização no espaço (CT') [23].

Uma outra técnica para a aquisição das três derivações unipolares do plano frontal foi introduzida posteriormente, por Emanuel Goldberger, para ultrapassar o problema associado às baixas amplitudes dos sinais adquiridos com as derivações unipolares (comparativamente com as derivações bipolares). Conforme ilustrado na Figura 2.17 [20,21], a solução para aumentar a amplitude de tais sinais passou pela remoção da resistência entre o elétrodo

posicionado no membro a ser medido e o CT. Ao conjunto das derivações obtidas com esta nova técnica deu-se o nome de derivações unipolares amplificadas, a cujas designações se atribuiu como prefixo a letra "a": obtendo-se portanto as designações aVR, aVR e aVF.

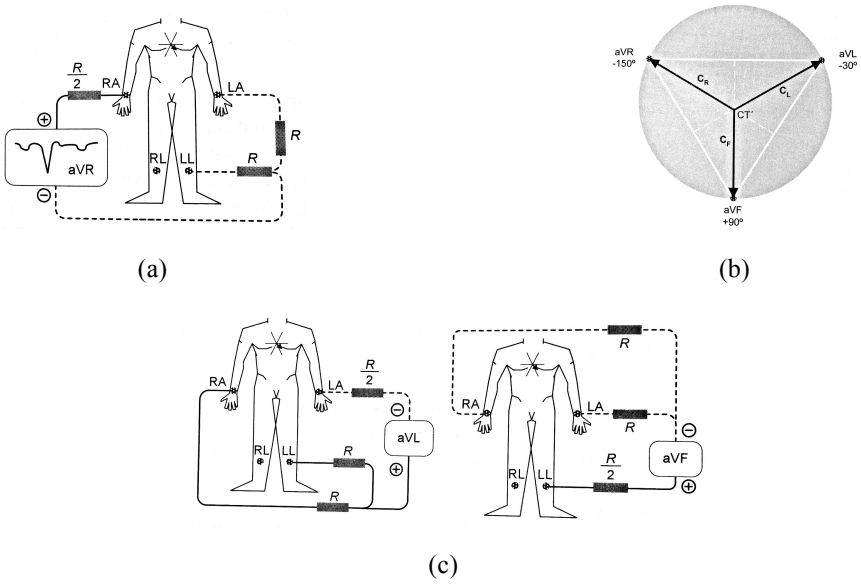


Figura 2.17 – Sistema de derivação proposto por Goldberger: (a) Exemplo da obtenção da derivação aVR pela desconexão do elétrodo do braço direito; (b) Representação das derivações unipolares aumentadas; e (c) Ilustração para obtenção das restantes derivações aVL e aVF [20].

Com o propósito de medir potenciais perto do coração, foi proposto o plano horizontal, no qual os elétrodos são aplicados em seis posições distintas do tóráx (

Figura 2.18), a partir das quais se obtêm as derivações unipolares pré-cordiais de Wilson (V_1, V_2, V_3, V_4, V_5 e V_6). Mais especificamente, os elétrodos V_1 e V_2 ocupam o quarto espaço intercostal, do lado direito e do lado esquerdo do esterno, respetivamente. Por sua vez, o elétrodo V_3 ocupa a posição entre V_2 e V_4, enquanto o posicionamento de V_4 é feito no cruzamento entre o quinto espaço intercostal e a linha média clavicular esquerda. Relativamente ao elétrodo V_5, este localiza-se entre V_4 e V_6, ao mesmo nível horizontal de V_4 mas na linha anterior axilar esquerda. Finalmente, o elétrodo V_6 ocupa o mesmo plano horizontal de V_4 mas na linha média axilar esquerda [18,24,25].

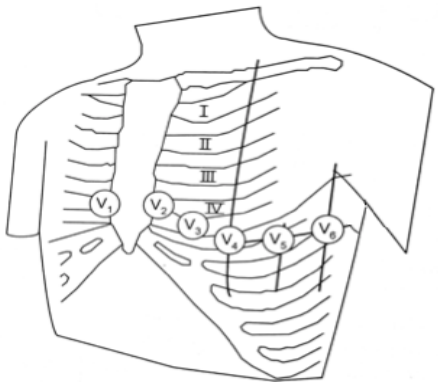


Figura 2.18 – Sistema de derivação unipolar precordiais do plano horizontal de Wilson [25].

2.5 Eletromiograma (EMG)

O eletromiograma (EMG) pode ser adquirido superficialmente, ao nível da pele, ou com elétrodos em agulha, diretamente inseridos no músculo alvo. As desvantagens do EMG superficial é que apenas podem ser medidos músculos superficiais e a leitura apresenta pouca resolução espacial. Além disso, o facto dos sinais de EMG serem compostos faz com que possam ser afetados por efeitos diversos, nomeadamente propriedades musculares, especificidades anatómicas e fisiológicas locais, interações ao nível do controlo do sistema nervoso periférico e os efeitos inerentes à instrumentação de aquisição. Os elétrodos em agulha podem ser monopolares ou multipolares e permitem ler sinais de EMG a partir de músculos profundos. O EMG é uma ferramenta clinica útil, pois é extensivamente utilizado no diagnóstico de problemas neurológicos e neuromusculares. Além disso, novas aplicações utilizando o EMG estão em estudo, nomeadamente em estudos sobre problemas de controlo de postura e marcha, bem como em aplicações biomecânicas envolvendo próteses [26].

2.6 Conclusões do Capítulo

Foram introduzidos os conceitos de variáveis fisiológicas e seus métodos de medição. Apresentou-se a anatomia do cérebro e do coração humano, servindo de apoio a introdução das medidas fisiológicas.

A obtenção de sinais EEG é muito complexa devido a sua baixa amplitude, a qual permite que o sinal seja envolvido pelo ruído do próprio sistema de instrumentação. O posicionamento dos elétrodos para a aquisição deste tipo de sinais é baseado no sistema internacional chamado de 10-20, o qual abrange todas as regiões de importância do marco ósseo do crânio humano. O estudo dos sinais EEG é baseado em ritmos, os quais dependem de vários fatores, como a frequência, amplitude, idade do individuo, o estado de alerta, entre outros.

Em relação aos sinais ECG viu-se que existem diversos métodos de obtenção de eletrocardiogramas que são baseados no sistema de derivações-padrões considerando que o coração é o centro de uma esfera concêntrica, sendo este um método não invasivo e muito simples de se aplicar.

Finalmente estudou-se o eletromiograma, que pode ser obtido superficialmente a nível da pele. No entanto este método não oferece grandes resultados, pois apresenta pouca resolução espacial, devido a diversos fatores. A substituição deste método implica a utilização de micro-agulhas que permitem ler diretamente no músculo que se pretende obter o sinal fisiológico. Esta técnica é de difícil aplicação neste trabalho, pois não se dispõem das condições, nem de pessoal qualificado para a realização dos mesmos.

3 Instrumentação Médica e Dispositivos de Medição Fisiológica

Os sinais biológicos, e em especial os sinais EEG, são medidos como potenciais elétricos, vulgarmente, designados por tensões. Estes sinais são fracos, contaminados pelo ruído e por sinais de interferência. Para medir os sinais de EEG, ECG e EMG são necessários amplificadores especiais, os quais conseguem detetar este tipo de sinais com alta-fidelidade. A primeira parte deste capítulo irá discutir alguns fatores e desafios a ter em conta quando se está a trabalhar na área das medições fisiológicas. Seguidamente são apresentadas algumas aplicações dos dispositivos de gravação fisiológica. No final do capítulo são apresentados alguns dispositivos que se encontram disponíveis no mercado atualmente, destacando as suas principais caraterísticas, vantagens e desvantagens. De referir que neste capítulo é dado um maior enfoque aos sinais do EEG pois são os mais exigentes de se obterem e os mais importantes neste projeto.

3.1 Sistemas de Instrumentação Médica

No projeto de um sistema de instrumentação médica alguns fatores devem ser tidos em consideração: o intervalo de valores do sinal de entrada, a largura de banda e o ruído, entre outros, desempenham um papel fundamental no desenho de um amplificador de instrumentação. Para a captura de sinais EEG, é necessário um ADC com uma resolução suficientemente elevada para medir com precisão o sinal (na faixa de 10 μV a 100 mV com frequências de 0,1 a 70 Hz). Na obtenção de sinais biológicos fatores ambientais, como a temperatura e a humidade, colocam restrições sobre os dipositivos a serem utilizados. Além disso a gama de alimentação e a sua frequência (50 ou 60 Hz) cria ruído variável, à frequência da linha de alimentação, que deve ser filtrado ou no pós-processamento ou através da filtragem por hardware.

3.1.1 Desafios

Esta subsecção vai apresentar uma introdução sobre os desafios na medição do EEG e as exigências que se colocam no desenho do hardware [27].

Impedância da Pele: A pele humana atua como uma impedância na medição do sinal EEG. Uma vez que esses sinais são muito fracos, para medir o sinal sobre o couro cabeludo, a impedância deve ser monitorizada e reduzida. Primeiro a pele deve ser limpa com álcool, e a colocação de um gel permite reduzir a impedância de contato entre a pele e o elétrodo. Note-se que a impedância da pele varia entre 10 kΩ e os 100 kΩ e deve ser reduzida para um valor abaixo dos 5 kΩ para que seja possível gravar os sinais do EEG com alta fidelidade.

Sinal em Modo Comum: Os sinais de EEG são sinais fracos e são significativamente afetados por sinais em modo comum. A impedância de entrada entre os elétrodos contribui

ainda mais para um sinal em modo comum elevado. Por este motivo, o amplificador requer uma razão de rejeição do modo comum elevada.

Interferências da Linha de Alimentação: O fornecimento de energia elétrica produz uma grande quantidade de interferências à frequência da linha de alimentação, que é de 50 ou 60 Hz, dependendo da região. Com conexões de boa qualidade e blindagem ativa este artefacto pode ser altamente minimizado.

Artefacto Muscular: Os movimentos musculares produzem os sinais dos eletromiogramas (EMG), cuja amplitude vária de 100 até 1000 μV, sendo que a maior parte da energia está concentrada na faixa de 50 até 150 Hz. Este sinal interfere significativamente com o sinal EEG. No entanto, com um filtro passa-baixo simples na faixa dos 30 Hz pode-se eliminar a maior parte desses artefactos musculares. O processamento de sinal digital também pode ser utilizado, tanto em software ou hardware, para remover esses artefactos.

3.1.2 Requisitos

Com base nos sistemas de instrumentação e desafios da medição de EEG, os seguintes requisitos básicos para um amplificador de instrumentação podem ser estabelecidos [27]:

Ganho Elevado: O amplificador deve ter um elevado ganho para amplificar o fraco sinal de entrada, de modo a que este não seja afetado pelo ruído interno do amplificador.

Alta Impedância de Entrada: O amplificador deve ter uma alta impedância de entrada, para que proporcione um efeito de carga mínimo sobre o sinal EEG. O efeito de carga elétrica pode resultar em distorções, portanto, os amplificadores devem ter uma alta impedância de entrada, geralmente superior a 10 MΩ.

Baixa Corrente de Entrada: Os amplificadores operacionais necessitam de uma pequena corrente que flua através de suas entradas, para que os seus transístores internos estejam corretamente polarizados. Mas essa passagem de corrente deve ser limitada, para garantir a segurança humana, e deve ser inferior a alguns nA.

Ampla Gama de Entrada em Modo Comum: A entrada em modo comum é a tensão média nas entradas inversoras e não inversoras do amplificador (pinos de entrada). O amplificador deve ter uma ampla gama de entrada em modo comum de forma a que possa aceitar o sinal do EEG sem afetar a entrada.

Razão de Rejeição do Modo Comum (CMRR): É a capacidade de um amplificador rejeitar o sinal em modo comum, e amplificar o sinal diferencial. Quanto maior for o CMRR menor é o ruído de modo comum na saída.

Baixo Ruído: Cada amplificador produz algum ruído interno. Estes níveis de ruído devem ser inferiores a 10 nV. O ruído interno ocorre essencialmente devido ao ruído térmico.

3.1.3 Blocos Funcionais

O diagrama de blocos da Figura 3.1 mostra os blocos básicos que devem estar presentes no hardware de aquisição. Esses blocos são explicados de seguida [27].

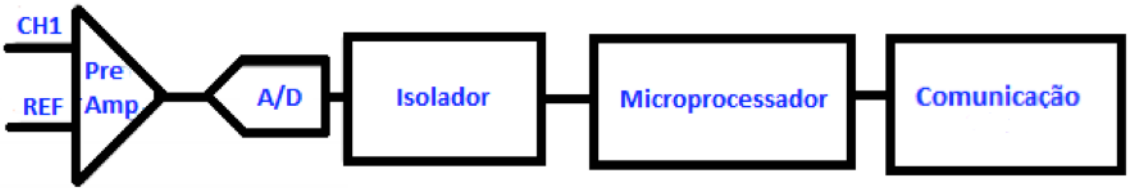


Figura 3.1 – Sistema de blocos funcionais de um sistema de aquisição de instrumentação médica.

Pré-amplificadores: Os pré-amplificadores devem ter as caraterísticas referidas na secção anterior.

Conversores Analógico-Digital (ADC): São usados para converter um sinal analógico numa sequência de *bits* digitais. Os sinais analógicos são mais suscetíveis ao ruído, pelo que geralmente são transformados para um formato digital. Normalmente a entrada é amostrada a intervalos de tempo específicos, T_s, a uma taxa $1/T_s = f_s$, onde f_s é a chamada frequência de amostragem. Um sistema de instrumentação moderno deve fornecer uma taxa de aquisição programável e dados em tempo real. Essas taxas de aquisição programáveis precisam de filtros de anti-*aliasing* variáveis. Esses filtros são caros e é necessário um para cada entrada analógica. O *aliasing* ocorre para sinais cuja frequência máxima, f, seja maior do que metade da frequência de amostragem, f_s, e essa componente de frequência mais elevada não poderá ser distinguida de uma componente de frequência mais baixa, ocorrendo interferências em $|f - nf_s|$ em que n, é qualquer número inteiro.

Isolamento: Qualquer corrente elétrica nos terminais de entrada de um amplificador pode resultar num choque elétrico para o utilizador. Para evitar este problema, o amplificador deve ter um circuito de isolamento da alimentação, de modo a que a corrente através do elétrodo possa ser mantida a níveis seguros e para que qualquer artefacto gerado pela corrente possa ser minimizado. Geralmente o circuito de isolamento é constituído por três isolamentos diferentes, o isolamento digital, da fonte de alimentação analógica e do fluxo de dados digitais.

Processamento de Dados: Permite controlar e processar o fluxo de dados da conversão digital, produzido pelos ADC. A aplicação de controlo deve ser desenvolvida para enviar, usar e armazenar o fluxo de *bits* digitais. Normalmente microprocessadores, microcontroladores ou processadores digitais de sinais são utilizados para esse fim.

Comunicação: Para processar e visualizar os dados digitais é importante enviar os mesmos para um PC, onde estes dados possam ser facilmente analisados e visualizados. Para esse fim muitas vezes utiliza-se a comunicação RS232 ou RS485. Hoje em dia, a maioria dos computadores substituem este tipo de comunicação por ligações USB, *Bluetooth* ou *wireless*.

3.2 Aplicações

O EEG é uma das mais antigas e mais utilizadas formas de obtenção de sinais fisiológicos médicos. O EEG tem uma resolução temporal na ordem dos milissegundos, o que é muito melhor do que as dos seus equivalentes, quer a ressonância magnética ou a tomografia computadorizada. Algumas das utilizações do EEG são mencionadas a seguir [23, 27]:

Aplicações clínicas: É amplamente usado em hospitais e clínicas. Algumas das aplicações incluem a monitorização da profundidade da anestesia, e o diagnóstico da morte encefálica e do coma. Também são utilizados para a monitorização de pacientes em cuidados intensivos, localização de convulsões, monitorização da epilepsia e diagnóstico de outras doenças neurológicas graves.

Pesquisa: Além do uso do EEG em ambientes clínicos e hospitalares, o EEG é também amplamente utilizado em diferentes disciplinas para fins de pesquisa, nomeadamente, em estudos de investigação realizados em seres humanos e animais.

Engenharia Biomédica: Nas áreas da engenharia biomédica, o EEG é muito utilizado para a pesquisa e desenvolvimento de algoritmos, pesquisa de sistemas de *feedback* para próteses, etc. A maioria dos trabalhos envolvem estudos da previsão do comportamento em animais, por exemplo utilizando ratos epiléticos. Estudos de EEG podem ter várias aplicações em dispositivos de reabilitação.

Investigação em Psicologia: O EEG também é utilizado para diversos estudos do estado cerebral no campo da psicologia. Esses estudos incluem o estudo de distúrbios do sono, estudos do desenvolvimento humano, estudos cognitivos e fatores humanos. O EEG ainda é utilizado para o estudo de várias perturbações mentais, incluindo depressão, desordens bipolares e outras doenças.

Neuro-Economia: O EEG permite estudar a tomada de decisões nos humanos. Esta pesquisa é geralmente limitada a ambientes académicos, em contraste com a de *neuro-marketing*.

Neuro-Marketing: É uma área de pesquisa recém desenvolvida que está a ser bastante utilizada para fins de *marketing* na indústria. É usada para estudar a resposta do cérebro, para entender do que os consumidores gostam ou não gostam. As grandes empresas, estudam o EEG dos consumidores quando expostos às suas publicidades e produtos.

3.2.1 Interfaces Cérebro-Computador (BCI) e Aplicações

Muitas pessoas com deficiências motoras graves sofrem de interrupções nas vias de comunicação entre os músculos e o cérebro. Alguns desses pacientes estão totalmente bloqueados e não podem mover qualquer parte de seu corpo. Ao longo das últimas três décadas muitos estudos têm demostrado que as interfaces cérebro-computador (*Brain–Computer–Interface* – BCI) são uma alternativa de comunicação, utilizando sinais de EEG recolhidos no cérebro. Uma BCI visa restaurar a via de comunicação para auxiliar/aumentar

as funções motoras humanas. As BCI foram desenvolvidas com base em observações dos principais ritmos do EEG. Embora essas interfaces tenham sido desenvolvidas inicialmente para fornecer meios alternativos de comunicação, agora estão a ser utilizadas numa infinidade de aplicações, as quais serão brevemente apresentadas a seguir [28,29,30,31]:

Tecnologia: A motivação inicial para o desenvolvimento de uma BCI era fornecer uma tecnologia que funcionasse como meio alternativo de comunicação para pessoas com doenças, incluindo a Esclerose Lateral Amiotrófica (*Amyotrophic Lateral Sclerosis* – ALS). Os investigadores têm estudado as BCI há mais de três décadas para desenvolverem dispositivos de apoio eficazes. Combinando elementos de robótica, têm vindo a ser desenvolvidas por exemplo cadeiras de rodas controladas a partir do cérebro.

Reabilitação: No campo da reabilitação e das próteses neuronais há pesquisas em curso para desenvolver terapias para restaurar o controlo motor aos indivíduos afetados. Muitos grupos de pesquisa estão a estudar a plasticidade neural e a encontrar formas de religação. Investigadores mostraram que um paciente paralisado aprendeu a regular os seus ritmos senso-motores para controlar a estimulação elétrica funcional (FES) dos músculos do braço e da mão, sendo capaz de executar tarefas simples, como segurar um copo. Muitos outros investigadores da área estão a realizar trabalhos sobre próteses neuronais e dispositivos de reabilitação o que hoje em dia continua a ser um campo muito promissor.

Diagnósticos de Cognição: Além do seu uso em tecnologias de apoio e reabilitação, as BCI foram desenvolvidas para auxiliar no diagnóstico clínico, influenciando e melhorando o funcionamento cognitivo. Usando *neuro-feedback* as BCI têm sido utilizadas na detecção de coma, ou na monitorização da atenção (por exemplo para a segurança do motorista).

Jogos: Alguns jogos com BCI foram desenvolvidos inicialmente para pacientes com problemas. Exemplos desses jogos incluem o Berlim BCI, que inclui um jogo simples (*Pacman*), controlado pela modulação de potenciais corticais lentos. Estes jogos não têm ficado longe do mundo dos jogos comerciais, e organizações comerciais, como a *NeuroSky* e a *Emotiv* desenvolvem este tipo de jogos.

3.3 Dispositivos Comerciais para Medidas Fisiológicas

A tecnologia tem evoluído a um ritmo impressionante, sendo que hoje em dia está-se cada vez mais dependente do mundo electrónico. No entanto a nível médico há muito que fazer nessa área. Nesta secção serão apresentados alguns dispositivos que permitem a gravação de sinais fisiológicos em humanos, que se encontram disponíveis no mercado e na instituição (UMa/M-ITI), e que auxiliaram o processo de desenvolvimento do protótipo a criar nesta dissertação.

3.3.1 Dispositivo *g.tec g.MOBIlab+*

O dispositivo *g.tec g.MOBIlab+* é um amplificador de sinais vitais de alta performance e de alta precisão juntamente com um sistema de aquisição/processamento, desenvolvido pela

empresa *g.tec Medical Engineering*. Este permite a investigação cerebral, da atividade muscular, dos movimentos dos olhos, da respiração, da resposta galvânica da pele e de muitos outros parâmetros fisiológicos e físicos. Devido às suas especificações técnicas e diferentes opções de software, este instrumento tornou-se um padrão para diversos campos de pesquisa, incluindo a neuro-psicologia, as ciências da vida, a investigação médica e o *biofeedback*/*neurofeedback* pesquisa BCI. Este dispositivo comunica através de uma ligação sem fios (*Bluetooth*) e dispõe de 16 canais, amostrados simultaneamente a 24 bits. Como grande vantagem tem-se a portabilidade do sistema e a sua grande fiabilidade na obtenção de sinais EEG. Como grande desvantagem surge o seu custo elevado, rondando os 8000 €, e ainda o facto que este dispositivo não permite a utilização de elétrodos passivos [32]. Na Figura 3.2 é apresentado este dispositivo.

Figura 3.2 – Dispositivo *g.tec MOBIlab+*.

3.3.2 Dispositivo *Enobio NE*

O dispositivo *Enobio NE* é um sistema sensor de eletrofisiológia, sem fios, que permite a gravação de sinais de EEG. O *Enobio NE* tem 8 canais e é ideal para aplicações fora do laboratório, pois vem integrado com uma interface muito intuitiva e poderosa que permite ao utilizador uma fácil configuração, gravação e visualização de dados de um EEG com 24 bits de resolução por canal, 500 SPS, incluindo espectrogramas e visualização 3D em tempo real com caraterísticas espectrais. Este dispositivo tem também integrado um acelerómetro triaxial e um cartão microSD para guardar os dados. A grande desvantagem deste dispositivo é o seu elevado custo, rondando os 5000 € [33]. Na Figura 3.3 pode-se visualizar o dispositivo *Enobio NE* com os seus elétrodos integrados.

Figura 3.3 – Dispositivo *Enobio NE*.

3.3.3 Dispositivo *ModularEEG*

Infelizmente os dispositivos de EEG comerciais são geralmente de custo muito elevado para se tornarem uma ferramenta para *hobby* ou um brinquedo. Apesar disso a comunidade *OpenSource*, decidiu criar um dispositivo que permita a gravação de sinais fisiológicos, nomeadamente EEG. O dispositivo *ModularEEG* é composto por dois amplificadores EEG, e uma placa de captura de sinal de 6 canais que se conecta a um PC através de um cabo série padrão. O custo do sistema completo é de 320 €. Apesar de este ser um projeto interessante, só permite neste momento a gravação de EEG, e com baixa resolução (8 bits). Como grande vantagem, por ser um dispositivo *OpenSource,* deve ser referido que apresenta diversas ferramentas de interação compatíveis (BCI) [34].

3.3.4 Dispositivo *PLUX BioSignals*

O dispositivo *PLUX* foi criado no seguimento da tese de doutoramento de Hugo Gamboa no Instituto Superior Técnico de Lisboa, pertencendo hoje a uma empresa focada na investigação na área da reabilitação, pioneira no desenvolvimento de sistemas de aquisição de biosinais sem fios aplicados na medicina, no desporto e na investigação. O dispositivo, também conhecido por *BioPLUX*, é uma unidade de aquisição sem fios que, quando ligado com sensores biomédicos, colocados em diferentes pontos do corpo humano, consegue monitorizar sinais como a força muscular, temperatura, batimento cardíaco, respiração, sudação, entre outros. Os resultados são mostrados em tempo real num dispositivo computacional, e facilmente interpretados pelo utilizador. O dispositivo possui 8 canais com uma resolução de 12 bits, sendo este factor uma pequena desvantagem do dispositivo, pois existem no mercado outros dispositivos com a mesma capacidade de medição mas com uma resolução mais elevada e com mais canais. Em contrapartida este dispositivo encontra-se disponível no mercado a partir de 1000 € na sua versão mais simples (só o dispositivo) e a partir de 3500 € com os seus módulos sensores (ECG, EMG e *Electrodermal* – EDA) [35]. Na Figura 3.4 encontra-se representado este dispositivo.

Figura 3.4 – Dispositivo *PLUX BioSignals* [35].

3.3.5 Dispositivo *BITalino BioSignals*

O *BITalino*, ao contrário das demais soluções existentes, é um *kit* de desenvolvimento biomédico de baixo custo, que permite criar diversos projetos, utilizando sensores ou ferramentas fisiológicas. Este dispositivo apresenta-se com uma estrutura "*Protosnap"*, o que permite programar a placa inteira ou separar cada parte para futuros projetos. O *BITalino* é programado através de uma API que inclui linguagens como o *Python*, *Java*, *Android*, entre outras. Este dispositivo está equipado com 9 módulos, nomeadamente os módulos microcontrolador (*Multipoint Control Unit* – MCU), *Bluetooth*, alimentação, EMG, medição da condutividade da pele (EDA), ECG, acelerómetro de três eixos, led e sensor de luz. Estedispositivo utiliza o microcontrolador *ATMega* 328 (o mesmo que utiliza o Arduíno UNO). Suporta seis entradas analógicas (4 entradas de 10 bits e 2 entradas de 6 bits), e ainda tem 4 entradas digitais. Como vantagens este dispositivo apresenta a sua versatilidade e uma basta gama de escolha e ferramentas para medições. Outra grande vantagem é que pode ser programado através de várias linguagens o que o torna multiplataforma. No entanto, este dispositivo não pode ser considerado um bom equipamento médico, pois a sua baixa resolução, e os poucos canais analógicos disponíveis, representam uma grande desvantagem deste dispositivo. Este dispositivo encontra-se no mercado a um preço que ronda os 200 € [36]. Na Figura 3.5, pode-se visualizar o dispositivo *BITalino*.

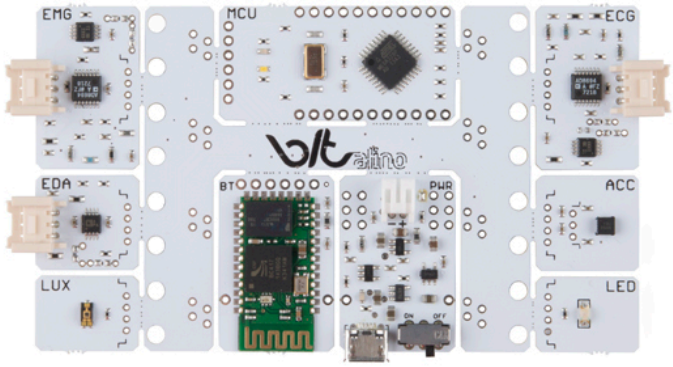


Figura 3.5 – Dispositivo *BITalino*, apresentando os seus diversos módulos [36].

3.4 Conclusões do Capítulo

Neste capítulo foram apresentados alguns desafios e requisitos que se devem ter em consideração quando se trabalha na área de instrumentação médica.

O sistema de blocos para a aquisição de sinais fisiológicos é complexo, começando pelos pré-amplificadores que devem ser capazes de amplificar as baixas amplitudes dos sinais e, os conversores A/D devem ser adequados de modo a não se perderem dados na passagem dos dados analógicos para digitais pois nas aquisições em tempo real são necessários filtros *anti-aliasing* variáveis.

Viu-se ainda neste capítulo que existem inúmeras aplicações relacionadas com a aquisição deste tipo de sinais, que englobam principalmente o ramo médico e outras áreas como o neuro-*marketing* e os jogos.

Finalmente foram apresentados alguns dispositivos de medição fisiológica, os quais podem ser visualizados na Tabela 3.1.

Tabela 3.1 – Comparação de alguns dispositivos de medição fisiológica existentes no mercado.

Dispositivo	Medidas Fisiológicas	Número de Canais	Resolução (*Bits*)	Tipo de Comunicação	Preço (€)
MOBIlab+	EEG	16	24	*Bluetooth*	8000
Enobio NE	EEG	8	24	*Bluetooth*	5000
ModularEEG	EEG	6	8	Série	320
PLUX	ECG/EMG/EDA	8	12	*Bluetooth*	3500
BITalino	ECG/EMG/EDA	6	10	*Bluetooth*	200

Atendendo à comparação apresentada na Tabela 3.1, pode-se concluir que atualmente no mercado existem diversas opções no que se refere a dispositivos de aquisição de medidas fisiológicas, no entanto nenhum dos dispositivos apresentados possui as características desejadas para o mesmo poder ser considerado um dispositivo de baixo custo com grande resolução e que permita a medição de diversas medidas fisiológicas, tais como EEG, ECG e EMG, num único dispositivo.

4 Dispositivo ADS1299

Os avanços da tecnologia e o desenvolvimento de soluções de baixo custo, nomeadamente de sensores de aquisição de sinais de baixa potência e de ADC ajudam muito à criação de instrumentos médicos e, assim, contribuem para um melhor diagnóstico. Este capítulo irá apresentar em detalhe o dispositivo escolhido para formar parte do hardware desta dissertação, tendo em consideração os dispositivos apresentados no final do Capítulo 3. Serão apresentadas as suas principais características, protocolos de comunicação, esquemas de configuração, programação e operação.

4.1 Analog Front-End

O ADS1299 é um dispositivo desenvolvido pela empresa *Texas Instruments* (TI) para servir de ligação entre sensores de medidas fisiológicas e um sistema digital (microcontrolador ou PC), o que é designado como sendo um *Analog Front-End* (AFE). Este dispositivo multicanal apresenta um baixo nível de ruído, com amostragem simultânea de 8 canais, a 24 bits, através de ADC do tipo delta-sigma (ΔΣ), com um amplificador de ganho embutido programável (*Programmable Gain Amplifier* – PGA), uma referência de tensão interna, e um oscilador *onboard* [37,38].

A empresa TI disponibiliza o ADS1299 integrado numa placa de desenvolvimento, a qual facilita o acesso aos seus pinos, de modo que os mesmos permitam medir um sinal fisiológico, e conseguindo desta maneira um baixo ruído na referência de entrada. Na Figura 4.1, pode-se visualizar a placa de desenvolvimento que inclui o AFE ADS1299.

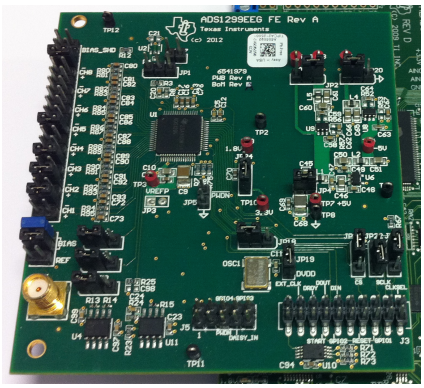


Figura 4.1 – Placa de desenvolvimento do AFE ADS1299.

O dispositivo ADS1299 fornece 8 canais (diferenciais ou não diferenciais) ligados a um multiplexador de entradas flexível, que permite que cada canal seja ligado de forma independente para diferentes tipos de sinais, nomeadamente, sinais de teste gerados internamente, sinais externos através da utilização de elétrodos, teste de temperatura, entre outros. Esses canais operam com taxas de amostragem entre 250 SPS e 16000 SPS, com uma resolução de 24 bits por canal, em simultâneo. A inclusão de um PGA em cada canal permite obter ganhos de até 24x, programáveis de forma independente para cada um dos canais presentes no ADS1299 [38].

Vários dispositivos ADS1299 podem ser conectados em cascata (*daisy-chain*), aumentando assim o número de canais disponíveis, até um máximo de 32 canais, ou seja, 4 dispositivos ADS1299 conectados entre si.

Segundo o fabricante, o AFE ADS1299 é o dispositivo que apresenta menor ruído no mercado dos dispositivos de obtenção de medidas fisiológicas de baixo custo (175€ aproximadamente). A placa de desenvolvimento ADS1299 vem acompanhada de uma outra placa DSP (*Digital Signal Processor*), MMB0, que permite a aquisição dos dados fisiológicos do ADS1299, através de uma interface desenvolvida pela empresa TI em *LabVIEW*, a qual será apresentada no Capítulo 5.

O dispositivo ADS1299 utiliza ADC com a arquitetura ΔΣ, sendo esta uma grande vantagem, pois essa arquitetura elimina a necessidade de ganho adicional e, idealmente, exclui a necessidade de filtragem antes da passagem para o domínio digital (*anti-aliasing*) [38]. Com taxas de aquisição de dados variáveis o custo de circuitos para diferentes ganhos e filtros *anti-aliasing* analógicos é elevado. A arquitetura ΔΣ elimina a necessidade de qualquer circuito adicional, reduzindo consideravelmente o custo.

O ADS1299 incorpora todas as características normalmente exigidas para a obtenção de sinais fisiológicos, nomeadamente o EEG, o ECG, o EMG, com uma grande fiabilidade e com um custo muito reduzido. Por estes motivos o ADS1299 AFE foi o dispositivo escolhido para se atingirem os objetivos definidos no início desta dissertação, pois era, o que mais se assemelhava aos dispositivos comerciais, apresentados no final do Capítulo 3. Nas secções seguintes serão apresentadas as caraterísticas elétricas do ADS1299, bem como, o seu protocolo de comunicação, as configurações essenciais para o seu normal funcionamento, a programação dos seus registos e ainda a sua operação [37, 38].

4.2 Caraterísticas do ADS1299

Esta secção contém informações sobre os elementos funcionais internos do ADS1299. As caraterísticas elétricas serão apresentadas em primeiro lugar, seguindo-se uma breve explicação dos blocos analógicos, expondo a arquitetura ΔΣ [38]. A Figura 4.2, mostra o diagrama funcional do ADS1299, no qual é indicada a divisão nos seus blocos principais: (a) Entradas dos amplificadores; (b) Opções de entrada do ADS1299; (c) Bloco de comunicação SPI; (d) Relógio do ADS1299; (e) Amplificador de polarização e referência interna e externa; (f) Tensão de referência interna e externa; (g) Amplificadores de baixo ruído programáveis (PGA); (h) Conversores ADC e ainda a indicação amarelo e verde das tensões de alimentação, o *ground*, o pino de START e o de RESET do dispositivo ADS1299.

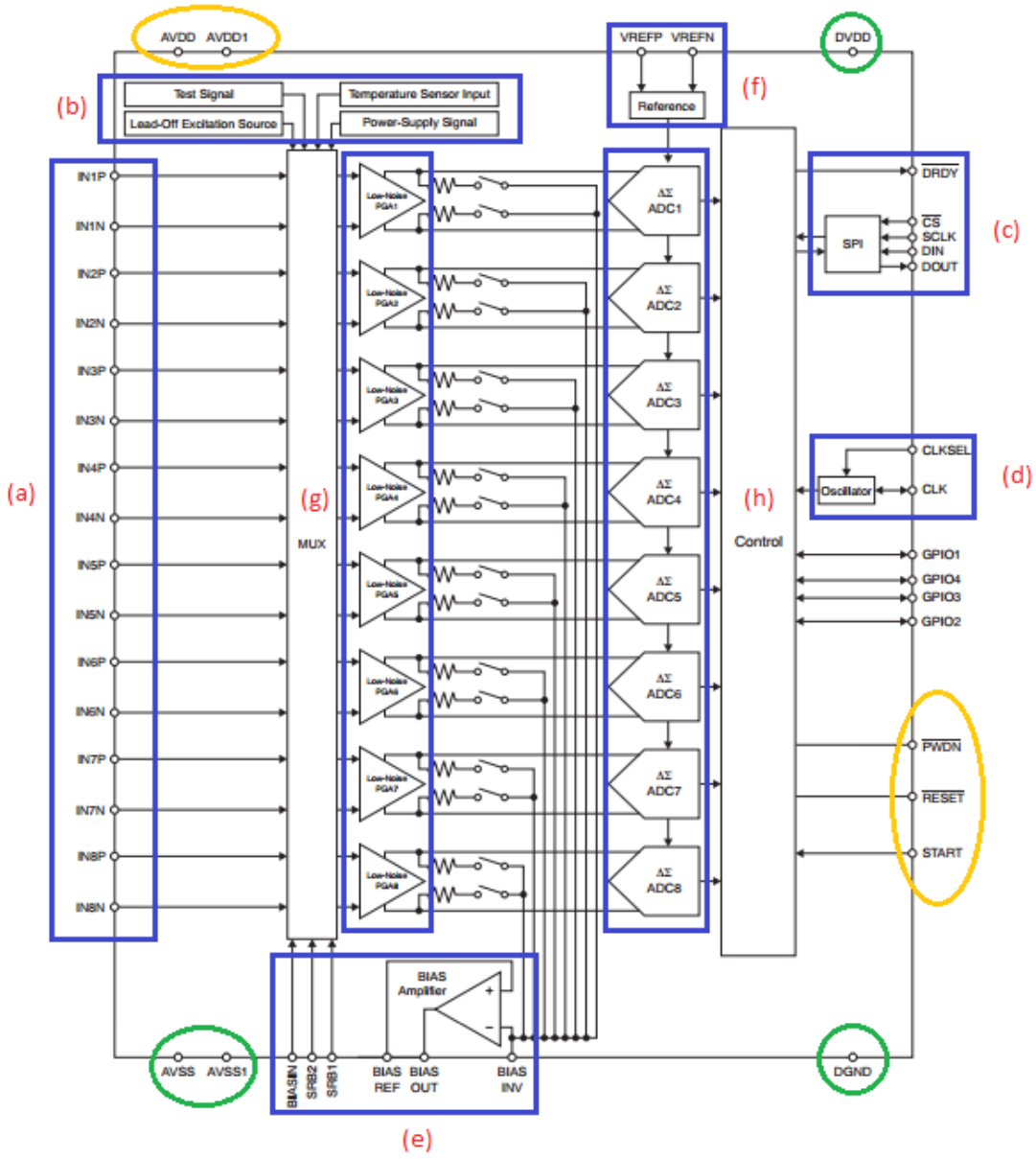


Figura 4.2 – Diagrama de Blocos do ADS1299 [37].

A descrição dos blocos da Figura 4.2, será apresentada nas diferentes secções que compõem este Capítulo.

4.2.1 Caraterísticas Elétricas

O ADS1299 é alimentado por tensões de 3,3 V e de 5 V, correspondendo a primeira à alimentação digital (DVDD menos DGND) e a segunda à alimentação analógica (AVSS menos AVDD). Os limites para essas tensões são, respetivamente, entre 4,75 V e 5,25 V e entre 1,8 V e 3,6 V. O dispositivo internamente funciona em baixa potência, com uma tensão de 4,5 V de referência interna, e tem uma faixa de temperaturas de funcionamento dos -40 °C aos +85 °C [37,38].

4.3 Comunicação com o ADS1299

O ADS1299 tem de ser programado para operar a uma velocidade de recolha e transmissão de dados específica. A única maneira de comunicar com o ADS1299 para programar e ler dados é via a Interface Periférica Série (*Serial Peripherical Interface* – SPI).

4.3.1 Comunicação SPI

A comunicação SPI é um padrão de comunicação usado para a comunicação com vários periféricos [37]. Na Figura 4.3, pode-se visualizar o esquema de ligação básico desta interface de comunicação, que pode ser visto como o envio de uma série de dados padronizados através de uma ligação síncrona. Nesse padrão os dispositivos comunicam no modo Mestre/Escravo (*Master/Slave)*, sendo o dispositivo *Master* que inicia a transferência de dados. A comunicação SPI utiliza quatro sinais lógicos:

SCLK (*Clock*): Relógio, que é emitido pelo *Master*.

MOSI (*Master Output Slave Input*): Linha para o envio de dados a partir do dispositivo *Master* para o *Slave*.

MISO (*Master Input Slave Output*): Linha para o envio de dados para o *Master* a partir do dispositivo *Slave*.

CS (*Chip Select*): No caso de existirem vários *Slaves* o *Master* pode escolher qual é o pretendido, colocando a entrada CS desse dispositivo no estado baixo (lógica negada).

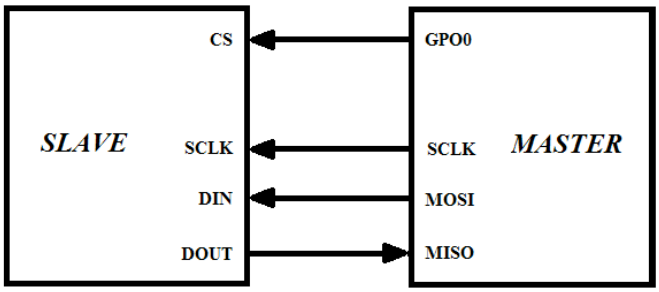


Figura 4.3 – Diagrama físico de uma ligação SPI.

Em termos de comunicação, durante cada ciclo de *clock* SPI, uma transmissão de dados *full-duplex* pode ocorrer, sendo que o *Master* envia um *bit* na linha MOSI e o S*lave* envia um *bit* na linha MISO que é lida pelo *Master*. Normalmente as transmissões envolvem dois registos de deslocamento de oito *bits*, um no modo *Master* e o outro no modo S*lave*, ligados uns aos outros em forma de um *buffer* em anel. Os dados são deslocados para fora com o *bit* mais significativo (MSB) em primeiro lugar, enquanto se dá a mudança de um novo *bit* menos significativo (LSB) no mesmo registo. Após esse registo ser deslocado para fora, o *Master* e o *Slave* trocam os valores de registo. Em seguida cada dispositivo tem esse valor e armazena ou executa a operação pretendida. Se existirem mais dados para serem trocados, os registos de deslocamento são carregados novamente com novos dados e o processo repete-se. O *Master* só pode selecionar um dispositivo *Slave* de cada vez. Quando vários dipositivos *Slaves* estão

presentes, os dispositivos são conectados em cascata, partilhando entre eles o canal de comunicação de saída (MISO), o canal de comunicação de entrada (MOSI) e o sinal de relógio (SCLK) [37].

4.4 Comandos para o ADS1299

O ADS1299 fornece uma forma flexível de programação e configuração do dispositivo, sendo a programação feita com base nos valores dos registos. Um *opcode* tem de ser enviado por meio da comunicação SPI para controlar e configurar a operação do dispositivo. Existem três tipos de comandos SPI, os quais são apresentados nas próximas 3 subsecções.

4.4.1 Comandos do Sistema

Estes comandos são comandos básicos do ADS1299, para o fazer reiniciar, iniciar as conversões, parar as conversões, acordar do modo de espera ou entrar em modo de espera. Estes comandos, também conhecidos por *opcodes*, são fixos e são apresentados de seguida [38]:

Acordar do Modo de Espera (*WAKEUP*): O envio deste *opcode* permite ao dispositivo sair do modo de espera. Qualquer comando subsequente deve ser enviado após 4 ciclos de relógio (T_{clk}).

Modo de Espera (*STANDBY*): O envio deste *opcode* coloca o ADS1299 no modo adormecido, exceto a referência.

Reiniciar (*RESET*): Este comando repõe todos os registos para os seus valores por defeito. São necessários 18 ciclos de T_{clk} para executar este comando, ficando limitado o envio de outros comandos durante esse tempo.

Iniciar Conversão (*START*): Este *opcode* inicia a conversão de dados. O software controla o pino START que deve ser colocado a *low*. Se as conversões já estão a decorrer este comando não terá efeito. Um período de 4 ciclos de T_{clk} deve estar presente entre o início e qualquer outro comando.

Parar Conversão (*STOP*): Este código de operação irá parar a conversão de dados.

4.4.2 Comandos de Aquisição de Dados

Estes comandos são usados para ativar a aquisição de dados por parte do ADS1299. Este oferece dois modos de ler os dados: o primeiro é o RDATA, em que é realizada apenas uma operação de leitura, o segundo consiste na leitura de dados em modo contínuo (RDATAC), sendo que a conversão contínua ocorre sem a necessidade de se enviar um outro *opcode*. Para parar a conversão contínua o *opcode* SDATAC deve ser enviado. Estes comandos são melhor explicados de seguida [37,38]:

RDATAC: Este comando coloca o ADS1299 na modalidade contínua de aquisição de dados, sem a necessidade de envio de *opcodes* subsequentes. Para este *opcode* trabalhar pela primeira vez o comando START deve ser previamente enviado. Este é o modo padrão do dispositivo quando é ligado e após um RESET.

SDATAC: Este *opcode* cancela o comando RDATAC.

RDATA: Este *opcode* é usado para a leitura dos dados de conversão de uma única amostra da base de amostragem. O comando START deve ser emitido anteriormente para o *opcode* RDATA funcionar.

4.4.3 Comandos de Leitura e de Escrita dos Registos

Estes *opcodes* são usados para programar o ADS1299, nomeadamente o *opcode* WREG permite escrever num registo e o comando RREG permite ler um registo. Ambos precisam de dois bytes para serem enviados. O primeiro byte contém o *opcode* da operação de comando e o endereço do registo. O segundo byte especifica o número de registos a serem lidos ou escritos menos um. Estes comandos são descritos a seguir [37, 38]:

RREG: Lê os dados do registo, sendo um *opcode* de dois bytes, que deverá ser seguido dos dados de saída. Os dois bytes do *opcode* contêm o *opcode* e o endereço do registo (001r rrrr), onde r rrrr é onde começa o endereço do registo do primeiro byte. O segundo byte especifica o número de registos a serem lidos menos um (000n nnnn), onde n nnnn é o número de registos a serem lidos menos um. Este comando pode ser emitido a qualquer momento.

WREG: Escreve os dados no registo, sendo um *opcode* de dois bytes, que deverá ser seguido dos dados de entrada. Esses dois bytes, contêm o *opcode* do comando e o endereço do registo (001r rrrr), onde r rrrr é o endereço inicial do registo no primeiro byte. O segundo byte especifica o número de registos a escrever menos um (000n nnnn), onde n nnnn é o número de registos a serem escritos menos um. Tal como o comando anterior, este pode também ser emitido a qualquer momento.

4.5 Programação do ADS1299

O ADS299 pode ser programado alterando os valores de seus registos internos, como foi mencionado na secção anterior. Esta secção irá destacar algumas das definições dos registos básicos que precisam de ser alteradas para um normal funcionamento do dispositivo.

As configurações básicas englobam a definição do modo de funcionamento, da frequência de amostragem, do ganho programável, do tipo de entrada, entre outros parâmetros do ADS1299. Para facilitar a TI organizou os registos do ADS1299, em vários grupos (cada um composto por 8 bits), os quais são definidos consoante o modo de funcionamento pretendido pelo utilizador. Esses grupos são apresentados e explicados de seguida [38]:

CONFIG1: Permite escolher o modo de operação, a fonte de relógio (*clock*) e a taxa de amostragem. A Tabela 4.1, mostra o mapa de registos para a configuração do registo CONFIG1.

Tabela 4.1 – Mapa de Configuração de registos CONFIG1 [38].

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
1	DAISY_EN	CLK_EN	1	0	DR2	DR1	DR0

Para uma melhor percepção será explicado cada um destes *bits* [37,38]:

Bit 7: Encontra-se sempre definido a "1" no dipositivo e não realiza qualquer função.

Bit 6 (DAISY_EN): Determina o modo em que o ADS1299 é ativado: definido a "0" (por defeito) para o funcionamento em cascata (*daisy-chain*) e "1" para a leitura de um único dispositivo. Na Figura 4.4, encontra-se o esquema de ligação em cascata de dois dispositivos.

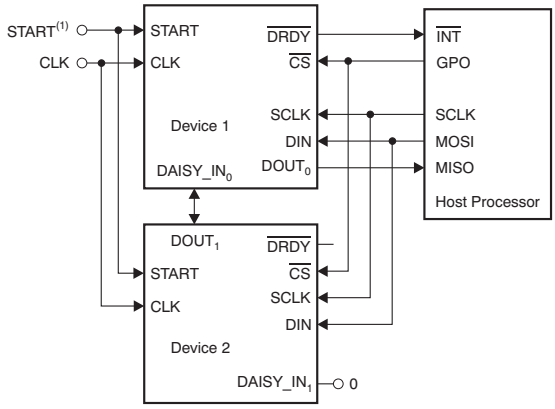


Figura 4.4 – Esquema de ligação de dois dispositivos em cascata [38].

Bit 5 (CLK_EN): Determina se o sinal do oscilador interno está ligado ao pino CLK, quando o pino CLKSEL=1. Deve ser usado "0" quando a saída do oscilador está desativada (por defeito), e "1" quando ativada. Uma tabela de verdade para estes dois parâmetros é mostrada na Tabela 4.2.

Tabela 4.2 – Relógio do sistema ADS1299 [38].

PINO CLKSEL	CONFIG1 BIT [CLK_EN]	RELÓGIO (*CLOCK*)	PINO CLK (STATUS)
0	X	Relógio Externo	Entrada: Relógio Externo
1	0	Oscilador Interno	----------
1	1	Oscilador Interno	Saída: Oscilador Interno

O relógio interno é ideal para um baixo consumo de energia, nomeadamente em sistemas alimentados por bateria. O bit CLK_EN é útil quando vários aparelhos são usados numa configuração em cascata. Quando o dispositivo é desligado, é recomendado desligar o relógio externo para economizar energia [37,38]. O relógio interno é de f_{CLK} = 2,048 MHz. A frequência do relógio externo deve variar entre 0,5 MHz e 2,25 MHz.

Bits [4:3]: Encontram-se sempre definidos a "1" e a "0", respetivamente, e não realizam qualquer função.

Bits [2:0] (DR): Permitem escolher a taxa de dados de saída do dispositivo. Esta taxa é calculada através de: $f_{Mod} = f_{CLK}/2$. Na Tabela 4.3, encontram-se expressas as taxas de amostragem com base na taxa de dados.

Tabela 4.3 – *Bits*[DR] que podem ser ajustados para se obter a taxa de amostragem desejada [38].

BITS	TAXA DE DADOS	TAXA DE AMOSTRAGEM
000	$f_{MOD}/64$	16 kSPS
001	$f_{MOD}/128$	8 kSPS
010	$f_{MOD}/256$	4 kSPS
011	$f_{MOD}/512$	2 kSPS
100	$f_{MOD}/1024$	1 kSPS
101	$f_{MOD}/2048$	500 SPS
110 (por defeito)	$f_{MOD}/4096$	250 SPS
111	Não se usa	Não definido

CONFIG2: O ADS1299 fornece opções para testes do ruído interno e de calibração e ainda da entrada de fontes externas. Estas opções podem ser configurados alterando os valores dos registos da configuração 2 (CONFIG2). A Tabela 4.4, mostra o mapa de registos para a configuração do registo CONFIG2.

Tabela 4.4 – Mapa de configuração de registos CONFIG2 [38].

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
1	1	0	INT_CAL	0	CAL_AMP0	CAL_FREQ1	CAL_FREQ0

Tal como para a configuração anterior, em seguida será apresentada a descrição de cada um dos bits que conformam o CONFIG2 [37,38]:

Bits [7:5]: Encontram-se sempre definidos a "110" no dipositivo e não realizam qualquer função.

Bit 4 (INT): É usado para selecionar a fonte de teste, "0" para sinais externos (gerador de sinais, elétrodos e diversos sensores) e "1" para a geração de sinais de testes gerados internamente (por defeito), servindo de calibração e para a medição do ruído do dispositivo.

Bit 3: Encontra-se sempre definido a "0" no dipositivo e não realiza qualquer função.

Bit 2 (TEST_AMP): A amplitude do sinal de teste pode ser definida com este *bit,* que determina a amplitude do sinal de calibração. Definido a "0" (por defeito), para 1×-(VREFP-VREFN)/2,4 mV e a "1" para o dobro desse valor.

Bits [1:0] (FREQ_TEST): Permitem definir a frequência do sinal de teste, e devem ser ajustados para "00" (por defeito) para $f_{CLK}/2^{21}$, a "01" para $f_{CLK}/ 2^{20}$, a "11" para as condições de corrente contínua. O padrão "10" não é usado.

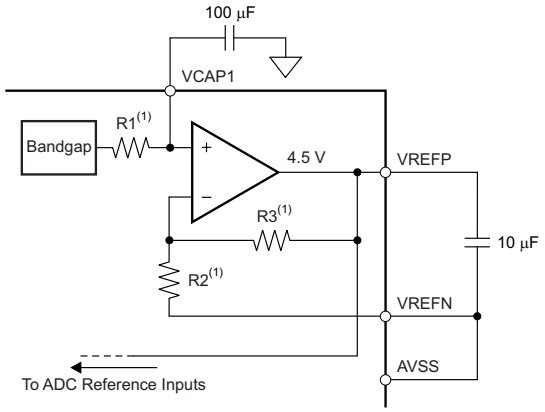
CONFIG3: A fim de se definirem os valores de referência para os ADC, os valores constantes do registo de configuração 3 (CONFIG3) devem ser escolhidos. A Tabela 4.5, mostra o mapa de registos para a configuração do registo CONFIG3.

Tabela 4.5 – Mapa de configuração de registos CONFIG3 [38].

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
PD_REFBUF	1	1	BIAS_MEAS	BIASREF_INT	PD_BIAS	BIAS_LOFF_SENS	BIAS_STAT

A descrição de cada um dos bits é apresentada a seguir [37,38]:

Bit 7 (PD_REFBUF): Determina o estado da referência interna, sendo definido como "0" (por defeito) para habilitar a referência externa e a "1" para habilitar a referência interna. A Figura 4.5, mostra um diagrama de blocos simplificado da referência interna do ADS1299. A tensão de referência de 4,5 V é gerada com respeito à tensão AVSS. Quando se usa a referência de tensão interna, deve-se conectar a tensão de referência negativa (VREFN) à tensão AVSS [23].



(1) For V_{REF} = 4.5 V: R1 = 9.8 kΩ, R2 = 13.4 kΩ, and R3 = 36.85 kΩ.

Figura 4.5 – Diagrama de blocos simplificado da geração da referência interna [38].

Os condensadores de limitação de bandas externas determinam o montante da contribuição do ruído de referência. Para sistemas de EEG de alta qualidade, os valores dos condensadores devem ser escolhidos de tal modo que a largura de banda seja limitada a menos de 10 Hz, de modo que o ruído da referência não domine o ruído do sistema [37].

Em alternativa, o *buffer* de referência interna pode ser desligado e uma tensão de referência positiva (VREFP) pode ser aplicada externamente. A Figura 4.6 mostra um circuito típico para gerar uma tensão de referência externa. Este estado também é usado para compartilhar referências internas quando dois ou mais dispositivos estão ligados em cascata [37,38].

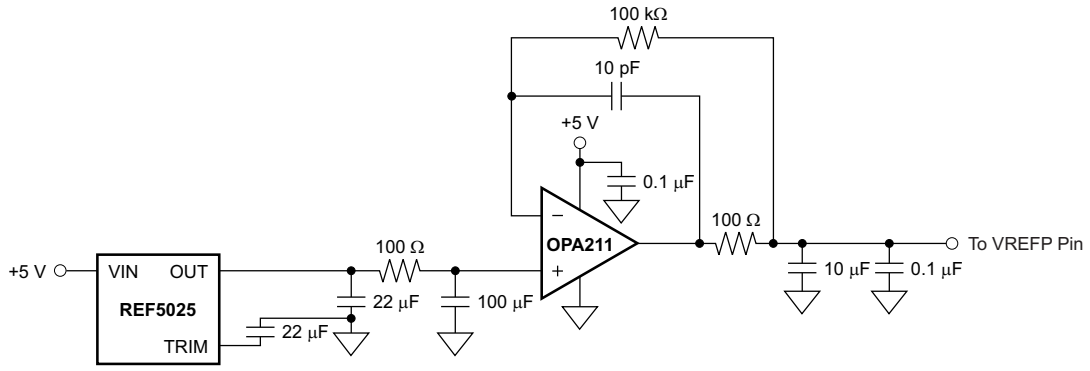


Figura 4.6 – Diagrama simplificado para a geração de uma referência externa [38].

Bits [6:5]: Encontram-se sempre definidos a "1" no dipositivo e não realizam qualquer função.

Bit 4 (BIAS_MEAS): Permite a medição do sinal de polarização. O sinal de polarização pode ser medido com qualquer canal. Definido a "0" (por defeito), indicando um estado de aberto, sendo colocado a "1" para definir um sinal de polarização interno (BIASIN), encaminhado para o canal que tem a tensão de referência (VREF).

Bit 3 (BIASREF_INT): Permite definir a fonte de sinal. Caso esta seja uma fonte externa o *bit* é definido a "0", e se for uma referência gerada internamente pelo dispositivo deve ser definido a "1" (por defeito). Esta referência interna é dada por (AVDD–AVSS)/2.

Bit 2 (PD_BIAS): Determina o estado do *buffer* do sinal de polarização. Este *bit* é definido a "0" (por defeito), o que determina que o *buffer* esteja desligado (*power-down)*, sendo colocado a "1" para ativar o *buffer*.

Bit 1 (BIAS_LOFF:SENS): Permite a ativação do sinal de polarização, e encontra-se a "0" (por defeito) para o estado desligado e a "1" para o estado ligado.

Bit 0 (BIAS_STAT): Permite escolher o estado do sinal de polarização. Este bit é definido a "0" (por defeito), o que indica que o sinal de polarização está conectado e a "1" indicando que o mesmo se encontra desligado.

CHnSET: Para alterar o modo de cada canal (ligado/desligado), o ganho do respetivo PGA e as configurações dos multiplexadores de entrada, têm de ser alterados os registos (CHnSET) de cada canal (onde CH1SET, corresponde à configuração do canal 1, CH2SET, ao canal 2 e assim por diante até ao número máximo de canais com que se está a trabalhar). A Tabela 4.6, mostra o mapa de registos para a configuração do registo de canal CHnSET.

Tabela 4.6 – Mapa de configuração de registos CHnSET [38].

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
PD1	GAIN12	GAIN11	GAIN10	SRB2	MUX12	MUX11	MUX10

A descrição de cada um dos bits é apresentada a seguir [37,38]:

Bit 7 (PD1): É definido a "0" (por defeito) no dispositivo, indicando a operação normal (canal ligado). Quando se pretende desligar esse canal (*power-down channel*) deve-se colocar esse *bit* a "1".

Bits [6:4] (GAIN): O ADS1299 têm 8 PGA de baixo ruído, um para cada canal. O dispositivo possui um potenciómetro interno (Figura 4.7) que permite a alteração do ganho do amplificador escolhido pelo utilizador.

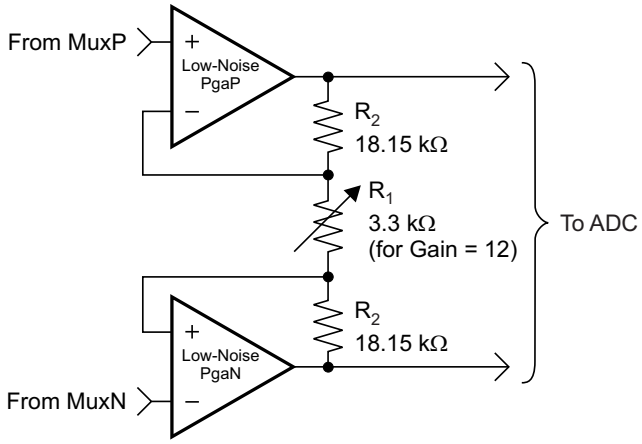


Figura 4.7 – Esquema de um Amplificador de Ganho Programável (PGA) do ADS1299 [38].

Para alterar os ganhos é preciso modificar estes *bits*. O nível de amplificação do PGA pode ser escolhido de forma independente para cada canal ou conjuntamente, consoante as necessidades do utilizador. Por defeito, este ganho é definido como 24, no entanto, pode ser configurado para se obterem ganhos de 1, 2, 4, 8, 12 ou 24. Na Tabela 4.7, podem-se visualizar os ganhos possíveis no dispositivo, apresentando-se a configuração dos bits para a sua correta escolha.

Tabela 4.7 – Configuração dos ganhos disponíveis no ADS1299 [38].

BITS	GANHO
000	1
001	2
010	4
011	6
100	8
101	12
110 (por defeito)	24
111	Não definido

De referir que para sinais EEG normalmente é utilizado um ganho de 24, devido as suas baixas amplitudes, e que para sinais ECG e EMG podem ser utilizados ganhos a partir de 8.

Bit 3 (SRB2): Permite ativar a referência do sinal de polarização e encontra-se definido a "0" (por defeito), indicando a não ligação. Para ativar a mesma o bit deve ter o valor de "1".

Bits [2:0] (MUXn): Ajudam a definir as configurações do multiplexador de entrada. Os multiplexadores de entrada do ADS1299 são muito flexíveis e oferecem várias opções de comutação de sinais configuráveis. A Figura 4.8, mostra o multiplexador de um único canal do dispositivo.

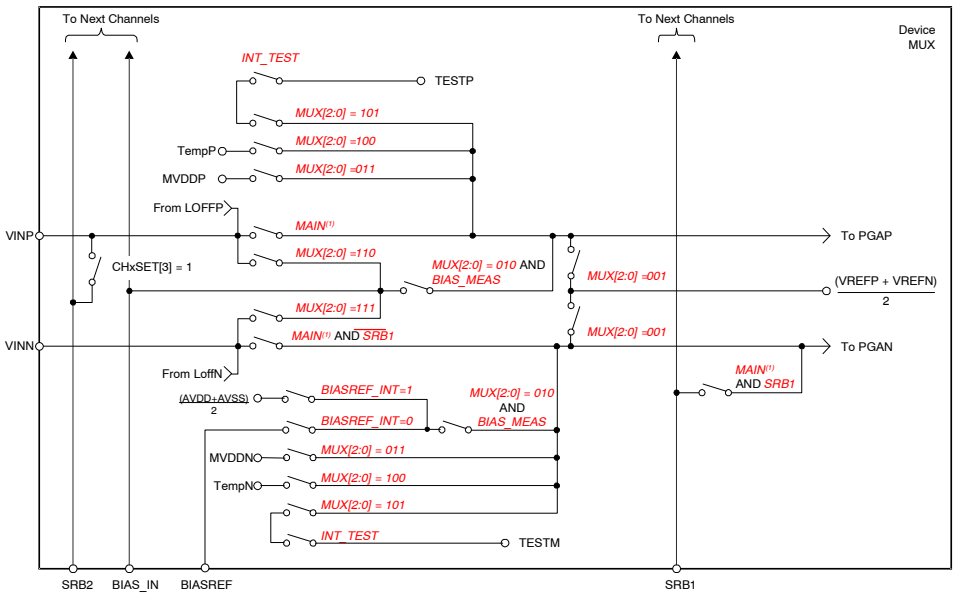


Figura 4.8 – Esquema com as opções do multiplexador de entrada analógica do ADS1299 [38].

Note-se que o dispositivo tem oito desses blocos, um para cada canal, sendo as entradas analógicas totalmente diferenciais e separadas (a tensão de entrada positiva é AINP e a entrada negativa é AINN. O sinal AINP–AINN pode variar entre $\pm V_{REF}$/gain onde V_{REF} é a tensão de referência). As entradas SRB1 (será descrita no registo seguinte), SRB2 e BIASIN são comuns a todos os oito blocos. Para definir o estado de ligação de cada interruptor do *Multiplexer* de entrada para cada canal é preciso definir os valores apropriados no registo CHnSET[2:0]. O ADS1299 fornece 8 opções de entrada diferentes, as quais são apresentadas na Tabela 4.8.

Tabela 4.8 – Opções das entradas analógicas disponíveis no ADS1299 [38].

BITS	ENTRADA ANALÓGICA
000 (Por defeito)	Elétrodo
001	Entradas curto-circuitadas
010	Condução da polarização
011	Fornecimento de MVDD
100	Sensor de temperatura
101	Sinais de testes gerados internamente
110	Polarização do elétrodo positivo
111	Polarização do elétrodo negativo

MISC1: Permite criar uma referência comum nos 8 canais (entradas negativas) do dispositivo. A Tabela 4.9, mostra o mapa de registos para a configuração do registo MISC1.

Tabela 4.9 – Mapa de configuração de registos MISC1 [38].

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	0	SRB1	0	0	0	0	0

A descrição dos *bits* deste registo é a seguinte [37,38]:

Bits [7:6]: Encontram-se sempre definidos a "00" no dipositivo e não realizam qualquer função.

Bit 5 (SBR1): Para escolher uma referência comum em todos os canais, o bit deverá ser "1" e para a não escolha deverá ser "0" (por defeito).

Bits [4:0]: Encontram-se sempre definidos todos a "0" no dipositivo e não realizam qualquer função.

4.6 Operação do ADS1299

A operação do ADS1299 envolve um conjunto de etapas sequenciais, sendo um fluxograma do funcionamento do hardware do ADS1299 mostrado na Figura 4.9. Em primeiro lugar todas as entradas digitais e analógicas devem ser desligadas (*power-down*). Após a ligação da fonte de alimentação, o sinal de relógio, deve ser fornecido no pino CLK, caso se deseje usar um relógio externo. Após um período de espera de 2^{16} ciclos de relógio (*clock*), o pino reiniciar (RESET) deve ser colocado em "1". Passado um segundo do impulso de reiniciar, e após um período de espera de 18 ciclos de relógio o dispositivo pode ser utilizado.

Inicialmente o comando SDATAC deve ser enviado. Em seguida, os registos devem ser programados através do envio do comando WREG, sendo que a referência deve ser definida. Posteriormente, pode ser enviado o *opcode* de início de leitura dos dados de conversão, usando o comando RDATAC. No ADS1299, por defeito, o pino reiniciar (RESET), encontra-se em "1" e o pino iniciar (START), encontra-se em "0". O fluxo de saída do dispositivo é de 27 bytes (24 bits de estado + 24 bits × 8 canais = 216 bits ou 27 bytes). O formato dos bits de estado (24 bits) é: (1100 + LOFF_STATP + LOFF_STATN + 4 bits de saídas e/ou entradas (GPIO) de registos). O ADS1299 fornece um total de quatro pinos GPIO disponíveis no modo normal de operação [37,38].

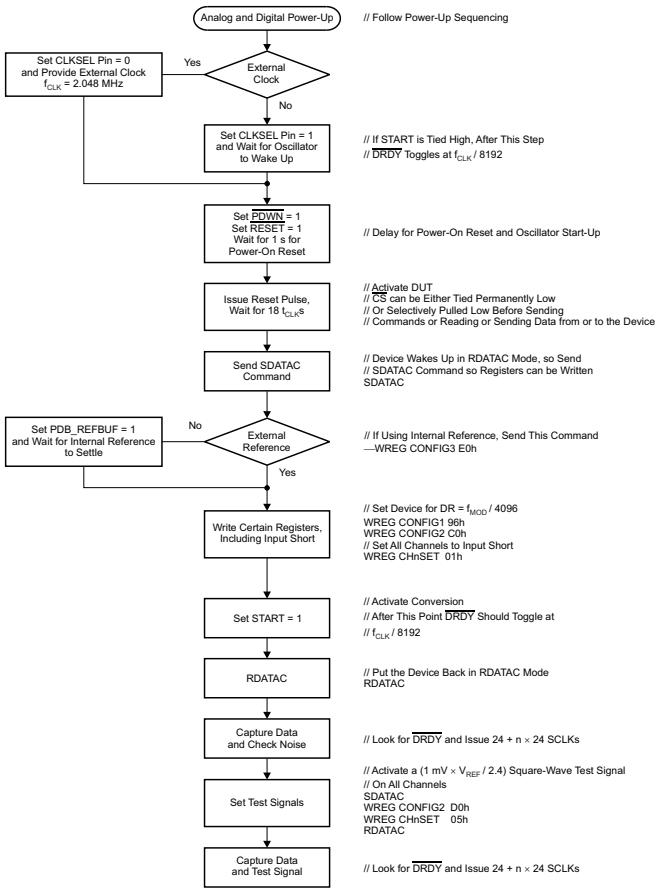


Figura 4.9 – Fluxograma do funcionamento do ADS1299 [38].

4.7 Conclusões do Capítulo

Neste capítulo foi apresentado o dispositivo que fez parte do hardware deste projeto, nomeadamente o AFE ADS1299, que é comercializado pela empresa TI. Inicialmente foram estudadas as suas características, métodos de comunicação e a respetiva configuração. Viu-se que o AFE ADS1299 é alimentado por duas tensões, 5 e 3,3 V, respetivamente, sendo este um factor de grande importância quando se pretende desenvolver um dispositivo alimentado com baterias. Uma vantagem deste dispositivo é que consegue comunicar com outros dispositivos por SPI, o que facilita a sua integração com outros equipamentos, os quais podem ser programados e configurados de uma forma flexível, com base nos valores dos seus registos. O AFE ADS1299 permite dois métodos de obtenção de dados fisiológicos, o primeiro em que o utilizador define um determinado número de amostras (SPS) e o segundo quando se pretende obter os dados de modo contínuo durante o tempo que o utilizador pretender, este último método também é chamado de *data streaming*.

Uma vez apresentado o dispositivo ADS1299 concluiu-se que o mesmo possui as características pretendidas para fazer parte do hardware do protótipo deste projeto, com um baixo custo (na ordem dos 200 €) com uma fiabilidade e resolução adequada.

5 Protótipo I

Neste Capítulo será abordado o dispositivo ADS1299 e a placa DSP que permite a aquisição de medidas fisiológicas. Inicialmente serão apresentadas algumas caraterísticas do módulo, e de seguida o software que permite visualizar as medidas fisiológicas, sendo apresentada a sua interface, as principais caraterísticas e funções. Finalmente será avaliado o seu desempenho, apresentando-se as suas vantagens e desvantagens.

5.1 Hardware: ADS1299 + DSP MMB0

No capítulo 4 foi referido que a placa de desenvolvimento AFE ADS1299 vem acompanha de uma DSP denominada de MMB0, a qual permite a aquisição de medidas fisiológicas. Esta DSP não é de uso exclusivo do AFE ADS1299, sendo também utilizada em diversos dispositivos desenvolvidos pela empresa TI. Na Figura 5.1, pode-se observar o conjunto dos módulos AFE ADS1299 + DSP MMB0.

O módulo é alimentado através de um transformador AC/DC, de 120 V a 220 V AC para 6 V DC. A placa ADS1299 comunica com a DSP via SPI e a comunicação do módulo com um PC é conseguida por meio de uma porta USB, sendo esta a única forma de comunicar, o que é uma grande desvantagem, pois não confere mobilidade ao sistema [37].

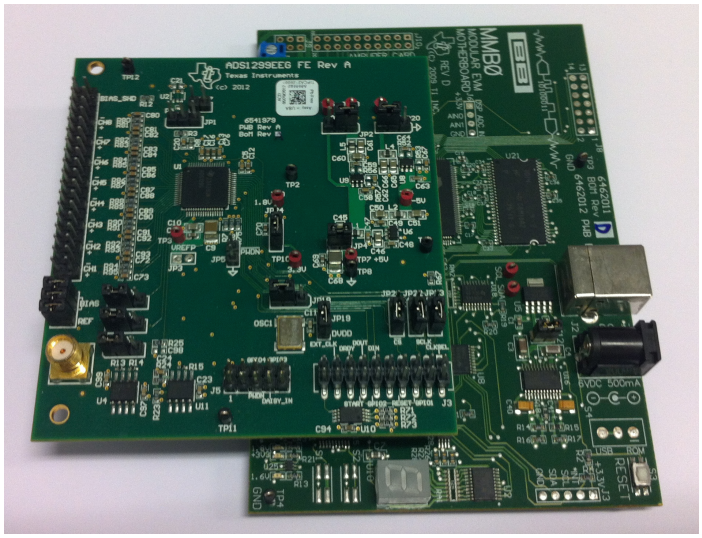


Figura 5.1 – Módulo desenvolvido pela TI: Placa AFE ADS1299 + DSP MMB0.

O módulo é composto por vários blocos funcionais os quais se interligam entre si, permitindo a obtenção de sinais fisiológicos, sendo este apresentado na Figura 5.2. Este sistema é composto por 8 blocos, dos quais 5 encontram-se na placa AFE ADS1299 e os outros 3 são comuns às duas placas.

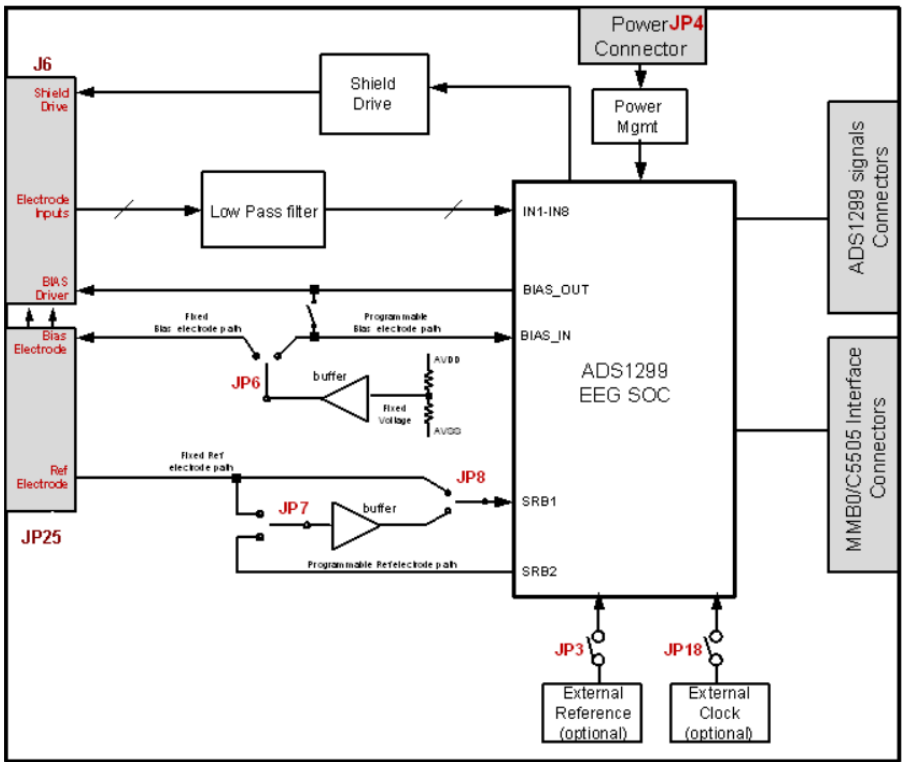


Figura 5.2 – Sistema de blocos funcional do módulo AFE ADS1299 + DPS MBB0 [38].

De referir que este sistema de blocos já foi explicado no capítulo 4, no entanto serão novamente apresentados aqui de um modo mais funcional:

Entrada de Sinal (*Electrode Inputs*): Composto pelos conectores do *jumper* J6, os quais correspondem aos 8 canais ADC.

Referência Interna dos Elétrodos (*Reference Electrode*): Composto pelos conectores do *jumper* J25 que se interligam aos JP6 (permite a ligação de um sinal de polarização), JP7 (referência interna negativa) ou JP8 (referência interna positiva).

Referência Externa (*External Reference*): Permite através da ligação do *jumper* JP3 a ligação a um sinal de referência externa.

Relógio Externo (*External Clock*): Possibilita adicionar um relógio externo ao módulo.

Alimentação (*Power Connector*): Correspondente ao bloco de alimentação do módulo.

Sinais (*Signals Connectors*) e Interface de Comunicação (*Interface Connectors*): Blocos que permitem a troca de dados via SPI.

De mencionar que cada bloco apresentando anteriormente é configurado através da colocação de *jumpers* ou da ligação de componentes, tais como, elétrodos, gerador de sinais, entre outros, que vão depender das necessidades do utilizador.

5.2 Software: Aplicação em *LabVIEW*

A empresa TI disponibiliza um software, que permite a visualização dos dados obtidos com o módulo apresentando anteriormente. O software consiste numa aplicação desenvolvida em ambiente *LabVIEW*, a qual é disponibilizada na versão executável (.exe) e na versão de código "aberto". A primeira versão deve ser instalada num PC, através de um processo simples e rápido, bastando clicar em "seguinte" até se finalizar a instalação. Após se ter sido instalado o software é necessário ligar o módulo ao PC e instalar os *drivers* USB da placa MMB0, os quais aparecem de forma automática no sistema operativo (SO), bastando também clicar em "seguinte" até se finalizar a detecção da porta USB. A segunda versão disponibiliza todos os ficheiros (sistemas de blocos, bibliotecas e classes) que compõem o software, no entanto com algumas restrições, já que alguns destes ficheiros encontram-se bloqueados, não permitindo o acesso ao seu código e outros encontram-se em código hexadecimal, dificultando assim ao programador a sua percepção, sendo esta uma grande desvantagem do software.

5.2.1 Interface da Aplicação

A aplicação fornece uma interface gráfica ao utilizador que pode ser visualizada na Figura 5.3, a qual se encontra dividida com os seguintes paneis: *About*, *ADC Register*, *Analysis, Save* e *Data Acquisition.*

De modo a perceber as funções e respetivo funcionamento de cada um destes paneis, os mesmos serão descritos seguidamente [38]:

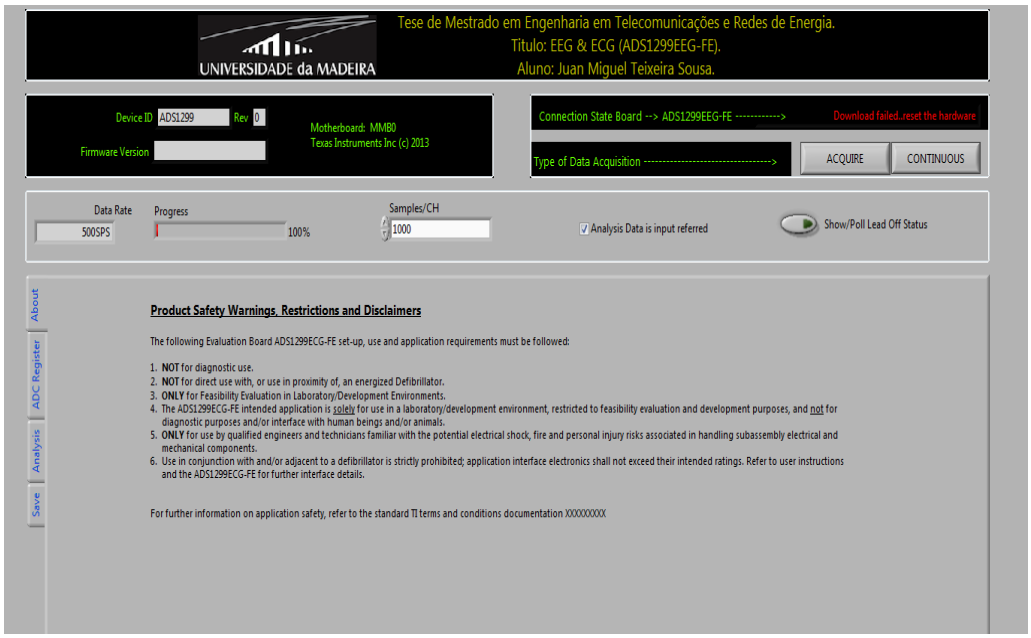
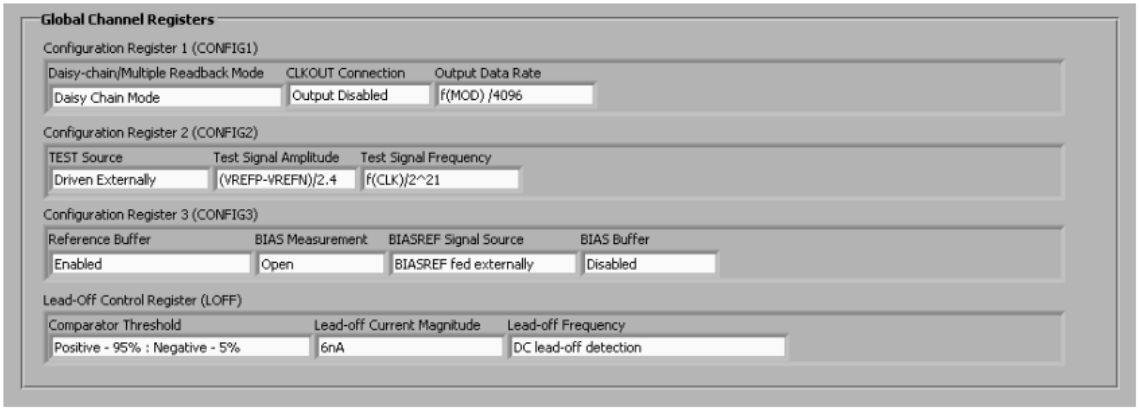


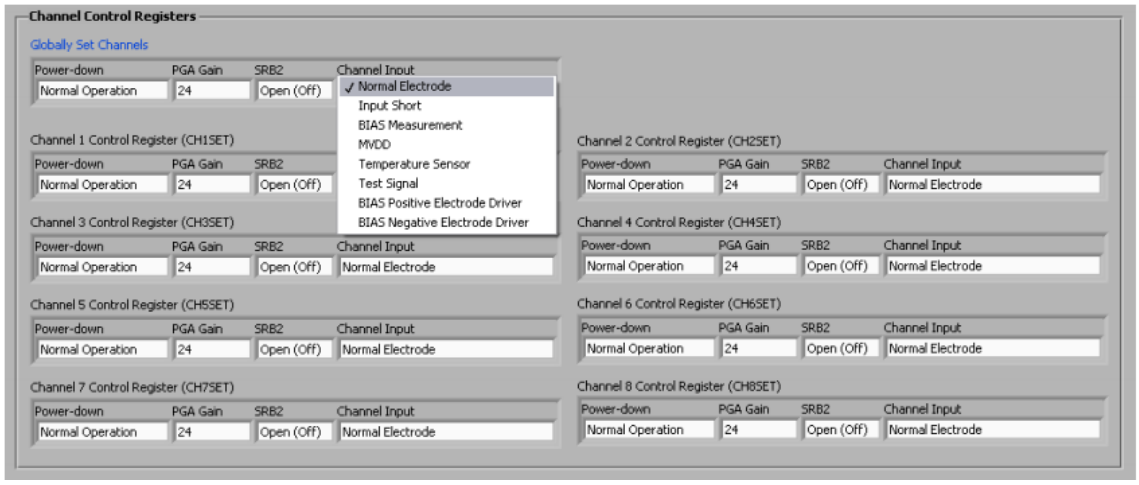
Figura 5.3 – Interface da aplicação em *LabVIEW* desenvolvida pela empresa TI.

Painel *About:* Este painel simplesmente contém alguma informação sobre a empresa e algumas recomendações para o uso do módulo.

Painel *ADC Register:* Permite visualizar e alterar os registos que compõem o dispositivo AFE ADS1299. Este painel, que pode ser visualizado na Figura 5.4, encontra-se subdivido em 3 partes, nomeadamente no *Global Channel Registers* (CONFIG1, CONFIG2 e CONFIG3), *Channel Control Registers* (CHnSET) e *Reference* (MISC1). De referir que todos estes registos aqui mencionados já foram apresentados no capítulo 4, pelo que não serão descritos novamente.



(a)



(b)

(c)

Figura 5.4 – Painel *ADC register* do AFE ADS1299: (a) *Global Channel Registers*; (b) *Channel Control Registers;* e (c) *Reference*.

Painel *Analysis:* Este painel possibilita a visualização dos dados, sobre a forma de gráficos, tendo o utilizador 3 opções de escolha: a visualização dos dados no domínio dos tempos, no domínio das frequências e ainda através de histogramas.

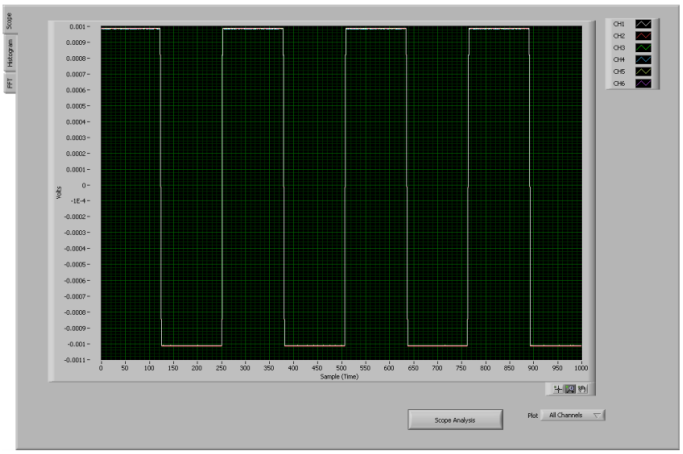


Figura 5.5 – Painel *Analysis* que permite a visualização dos traçados dos dados obtidos.

Painel *Save:* A aplicação também permite guardar os dados obtidos, num ficheiro de texto (.txt) ou excel (.csv). O utilizador pode ainda escolher quais são os dados que pretende salvar (os dados de um único canal, os dados de todos os canais em simultâneo ou de vários canais, respetivamente). Na Figura 5.6, pode-se observar o referido painel.

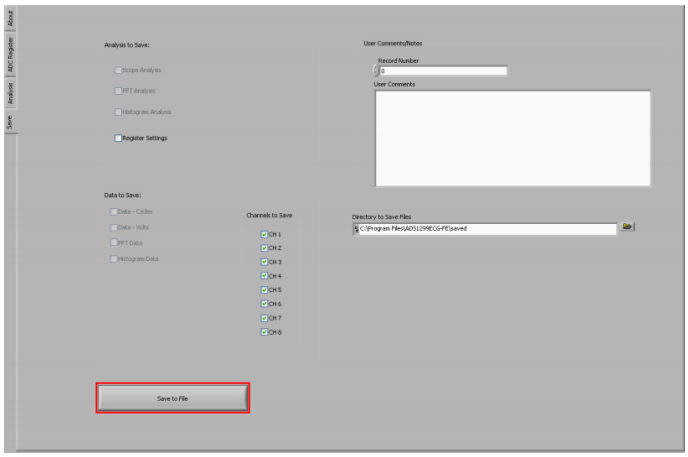


Figura 5.6 – Painel *Save* que permite guardar os dados obtidos.

Painel *Data Acquisition*: A aplicação permite dois modos diferentes de aquisição de dados, através da escolha de dois parâmetros: número de amostras por segundo (SPS) ou o número de amostras por canal (*Samples*/*CH*). O primeiro modo consiste numa única aquisição de dados (*Acquire*) baseado no parâmetro *Samples/CH* para um determinado SPS. O segundo modo é baseado nos mesmos parâmetros de escolha do primeiro, no entanto a aquisição de dados é feita de modo contínuo (*Continuous*), ou seja, faz uma primeira aquisição, baseada nos parâmetros escolhidos e volta a obter consecutivamente novas aquisições até que o utilizador decidir parar com a mesma.

Estes modos de aquisição podem ser escolhidos no painel superior da interface da aplicação, o qual se pode observar na Figura 5.7.

Figura 5.7 – Painel *Data Acquisition* que permite escolher o modo de adquisição de dados.

De mencionar que a aplicação tem ainda outras funcionalidades que não serão aqui apresentadas, pois não foram utilizadas no decorrer deste projeto e não foram consideradas importantes.

5.3 Configuração dos Registos

Os registos precisam de ser configurados pelo utilizador consoante os seus requisitos, sendo que os mesmos já foram descritos no capítulo anterior. De modo a poder avaliar o desempenho tanto do módulo de aquisição como da aplicação em *LabVIEW*, procedeu-se à configuração dos seus registos, de modo que fosse possível a posterior medição do ruído do protótipo I, verificando as ondas geradas internamente pelo dispositivo ADS1299. Foi também testada a obtenção de ondas geradas por um gerador de sinais de laboratório e, por último, a obtenção de sinais através de elétrodos de medição fisiológica.

Para se poder atender a todos os requisitos anteriores, foi decidido utilizar os canais em modo unipolar e em modo diferencial, utilizando-se para o primeiro modo as entradas positivas de cada canal, sendo curto-circuitadas as entradas negativas internamente através dos registos. Como foram escolhidas as entradas positivas de canal, a referência comum a todos os canais também foi escolhida positiva e gerada internamente (SRB1 ativo). Em relação ao ganho dos amplificadores de cada canal, foi decidido colocar todos com um ganho de 24x, tendo em conta que os sinais fisiológicos possuem amplitudes muito baixas. A opção de entrada de cada canal (*Channel Input*) dependeu do tipo de aquisição pretendida pelo utilizador, ou seja, no caso de se quererem obter sinais gerados internamente, deverá ser escolhida a opção *Test Signal*, caso fosse pretendida a obtenção de sinais externos era escolhida a opção *Normal Electrode* (por exemplo) e, assim sucessivamente para cada caso de obtenção.

De referir que no Anexo A se encontra a configuração de cada um dos registos e de cada um dos *jumpers,* no estado *default*, ou seja, na configuração por defeito cada vez que se reinicia o protótipo I.

5.4 Resultados Obtidos

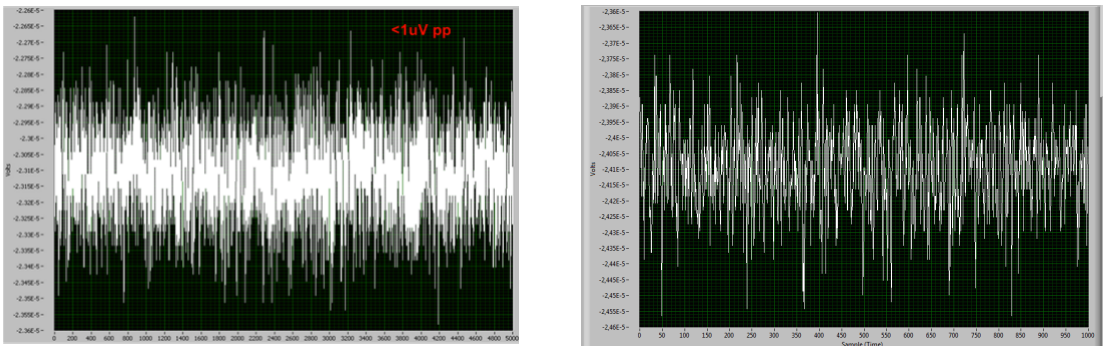
Nesta secção serão apresentados os testes e resultados obtidos com o protótipo I. Os testes foram divididos em 3 fases: a primeira consistiu na verificação do ruído do protótipo I, a segunda abrangeu a aquisição de ondas geradas internamente pelo dispositivo ADS1299 e a última fase consistiu na obtenção de ondas geradas por um gerador de sinais, permitindo avaliar o desempenho do protótipo I, sendo analisado se o mesmo permitia a correta obtenção de sinais ECG, EMG e EEG.

5.4.1 Medição do Ruído no Protótipo I

Como já foi referido ao longo deste projeto é fundamental conhecer o nível do ruído quando se trabalha na área de medições fisiológicas, sendo por isso que o primeiro teste com o protótipo I consistiu na medição do seu ruído.

O fabricante do protótipo I refere que o ruído envolvente no AFE ADS1299 é < 1 μV_{PP}, valor obtido para 10 segundos de aquisição de dados a 500 SPS e com um ganho de 24 (Figura 5.8-a). Para a obtenção e verificação destes valores realizou-se o teste com os mesmos parâmetros que o fabricante, sendo que para isto foi necessário curto-circuitar o canal 1, o qual é conseguido através da colocação de um *jumper* nos pinos de acesso ao canal na placa AFE ADS1299 (Anexo B). Existe ainda outro método de medição do ruído, que consiste em ligar o canal que se pretende medir ao pino AGND da placa AFE ADS1299, no entanto este procedimento não foi utilizado pois conduz aos mesmos resultados que o método anterior.

O resultado que se obteve com o protótipo I, pode ser visualizado na Figura 5.8-b. É de mencionar que o nível de ruído pode ser obtido para qualquer outro canal disponível ou para todos os canais em simultâneo.



(a) (b)

Figura 5.8 – Resultado do ruído obtido durante 10 segundos a 500 SPS com um ganho de 24: (a) Obtido pelo fabricante [38]; e (b) Obtido com o protótipo I.

Observando a Figura 5.8, pode-se dizer que se obtiveram resultados satisfatórios em termos de nível do ruído do protótipo I, pois este encontrava-se dentro da gama de valores que refere o fabricante, ou seja, um valor inferior a $1\mu V_{PP}$.

5.4.2 Obtenção de Ondas Geradas Internamente pelo ADS1299 com o Protótipo I

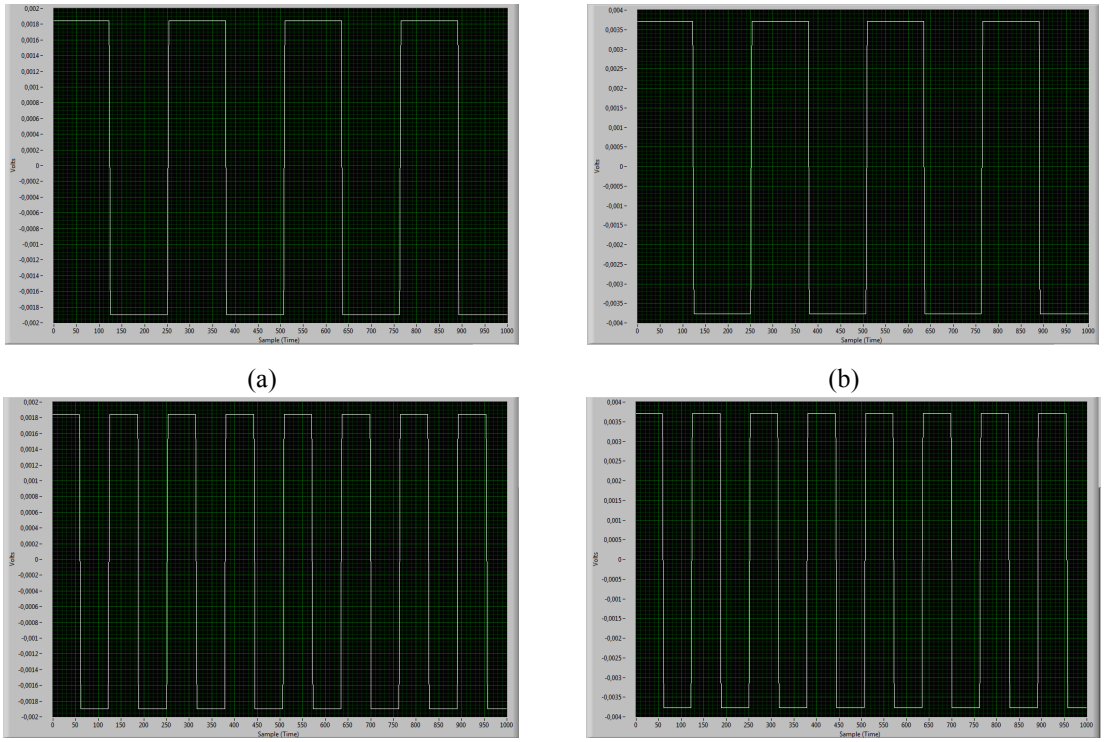
No capítulo 3 foi referido que o dispositivo ADS1299 consegue gerar ondas internamente. As ondas geradas são ondas quadradas, as quais podem ser configuradas em dois parâmetros:

Amplitude: (VREFP-VREFN)/2,4 mV ou 2x(VREFP-VREFN)/2,4 mV;

Frequência: $f_{CLK} = 2^{21}$ ou $f_{CLK} = 2^{20}$.

Atendendo ao apresentado anteriormente pode-se concluir que se conseguem obter 4 tipos de ondas quadradas diferentes.

Deste modo o segundo teste com o protótipo I consistiu na obtenção dessas ondas quadradas, sendo que foi necessário modificar quatro vezes os registos do ADS1299, uma vez para cada tipo de onda, sendo que essas modificações se encontram no Anexo C. As ondas foram obtidas a 250 SPS, durante um intervalo de 4 segundos, originando gráficos com 1000 amostras, os quais podem ser observados na Figura 5.9.



(a) (b)

(c) (d)

Figura 5.9 – Ondas geradas internamente pelo dispositivo ADS1299: (a) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (b) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (c) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$; e (d) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$.

Analisando os resultados da Figura 5.9, pode-se concluir que se conseguiram gerar as ondas quadradas internamente com o protótipo I, sem qualquer problema, utilizando unicamente a configuração dos registos internos. De referir que essas ondas não possuem qualquer ruído, como era de esperar, pois são geradas internamente pelo ADS1299.

5.4.3 Obtenção de Ondas Geradas por um Gerador de Sinais com o Protótipo I

Nesta secção pretendeu-se avaliar a capacidade de processamento e leitura dos ADC de cada canal do protótipo I. Para isso foram utilizados os dois modos de aquisição de dados apresentados anteriormente.

Com a ajuda de um gerador de sinais, disponível no laboratório, realizaram diversos testes ao protótipo I, no entanto só alguns serão apresentados aqui. Inicialmente optou-se por escolher o tipo de onda sinusoidal, e teve-se em consideração que a amplitude máxima suportada pelo ADS1299 é de 4,5 V_{PP}. Em relação à frequência foram escolhidos valores entre 1 e 200 Hz, visto a ser a gama de frequências dos sinais fisiológicos em estudo neste projeto.

Para a realização dos testes foi utilizado o canal 1, a qual foram ligadas as pontas do gerador de sinais, sendo que a ponta que corresponde à massa foi ligada ao pino de referência comum da placa AFE ADS1299 (Anexo D). De mencionar que os restantes canais foram desligados, através da aplicação em *LabVIEW*.

O primeiro sinal que se obteve com o protótipo I encontra-se representada na Figura 5.10, sendo este um sinal sinusoidal com amplitude de 0,001 V e uma frequência de 100 Hz, obtido com 16000 SPS. Este valor de amplitude é o mais baixo que se conseguia gerar com o gerador de sinais que estava disponível no laboratório.

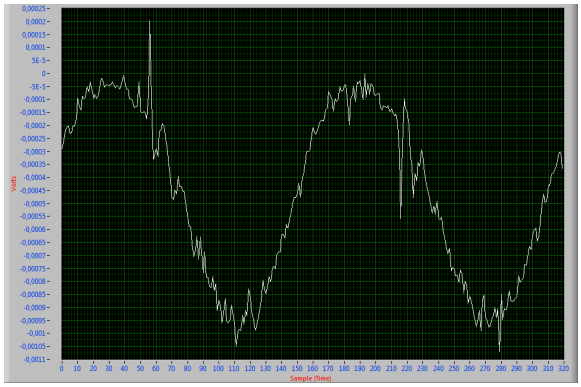
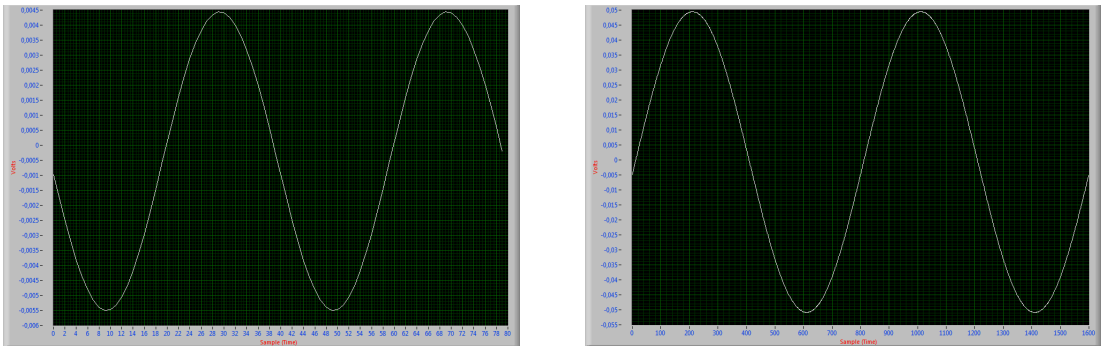


Figura 5.10 – Sinal sinusoidal de amplitude 0,001 V e frequência de 100 Hz, a 16000 SPS.

Observando a Figura 5.10, pode-se verificar a presença de ruído na onda tal deve-se a dois factores: o primeiro tem a ver com a resolução do gerador utilizado, tendo-se verificado que o mesmo não conseguia gerar satisfatoriamente sinais com a amplitude mais baixa. O segundo factor foi um pouco mais difícil de detetar, pois realizaram-se diversos testes, para várias amplitudes e várias frequências, e o ruído permanecia no sinal. Após alguns estudos, verificou-se que o problema vinha dos outros canais, ou seja, apesar de estes se encontrarem desligados na aplicação *LabVIEW*, foi necessário também desliga-los no hardware, não sendo isto o mencionado no *datasheet* do módulo. Para isso foram colocados *jumpers* em todos os canais que não estavam a ser utilizados, ficando estes curto-circuitados. Na Figura 5.11, podem-se visualizar algumas das ondas obtidas com o protótipo I, usando o modo de aquisição *Acquire.*



(a) (b)

Figura 5.11 – Ondas sinusoidais obtidas com o protótipo I, através do gerador de sinais: (a) Amplitude de 0,001 V e frequência de 50 Hz, a 2000 SPS; (b) Amplitude de 0,01 V e frequência de 50 Hz, a 2000 SPS;

Atendendo aos resultados da Figura 5.11, pode-se dizer que o protótipo I permite a obtenção de sinais de modo satisfatório, através deste modo de aquisição de um número predefinido de amostras.

Tendo sido analisado o primeiro modo de aquisição foi avaliado de seguida o segundo modo aquisição contínua, utilizando-se o procedimento anterior. Após alguns testes verificou-se que quando se tentava obter uma onda sinusoidal com o gerador de sinais para um determinado número de amostras por canal, as mesmas eram recolhidas inicialmente pela DSP e só após algum tempo é que eram disponibilizadas ao utilizador através da aplicação, existindo assim um intervalo de tempo entre a recolha e a visualização dos dados. Este modo de funcionamento fazia com que alguns dados fossem perdidos, causando ondas sinais sinusoidais com desvios de fase ou incompletas. Quando a aplicação estava a obter os dados do gerador, a interface ficava de cor cinzenta e cada vez que eram disponibilizados os novos dados era realizado um *refresh* à interface da aplicação, eliminando os dados anteriores.

No *datasheet* da placa AFE ADS1299 e da placa DSP MMB0 era referido que o módulo conformado por estas placas conseguia a obtenção de dados em tempo real, sendo também mencionado que a aplicação desenvolvida em *LabVIEW* é que não permita tal facto. Com base nessa informação foi tentado modificar esta aplicação, com o intuito de contornar o problema. No entanto essa tentativa não teve sucesso.

Após algum tempo, a TI atualizou o *datasheet*, sendo nessa altura referida a existência de uma limitação a nível do hardware, nomeadamente da placa DSP MMB0, que não permitia a aquisição de dados em tempo real, sendo esta uma grande desvantagem.

Devido a esta limitação do protótipo I, conclui-se o mesmo não cumpria com os objetivos iniciais deste projeto, pelo que foi necessário desenvolver uma outra solução, a qual será apresentada no próximo capítulo.

5.5 Conclusões do Capítulo

Neste capítulo foi apresentado o módulo de desenvolvimento da TI, o qual foi denominado de protótipo I, e o seu software em *LabVIEW*, que permite a obtenção e visualização de sinais fisiológicos.

Inicialmente foram realizados diversos testes, de elevada importância quando se trabalha na área de sinais fisiológicos, entre os quais, a verificação do nível de ruído, que se verificou que se encontrava adequado. Também foi testada e verificada satisfatoriamente a capacidade do protótipo I de gerar internamente ondas, que dependiam da configuração de dois parâmetros, nomeadamente, da amplitude e da frequência.

Posteriormente foi utilizado um gerador de sinais disponível no laboratório, ligado a um dos canais do protótipo I, de modo a verificar a capacidade de processamento deste tipo de sinais, tendo-se verificado que no modo de aquisição *Continuous* não se conseguiam obter os dados em tempo real, causando a perda de dados.

Concluiu-se então neste capítulo que o Protótipo I, não satisfazia os requisitos iniciais deste projeto, sendo que no próximo capítulo será desenvolvida e apresentada uma solução.

6 Protótipo II

É essencial para um bom dispositivo de medidas fisiológicas que este permita a visualização dos seus dados em tempo real. Neste capítulo é apresentado o desenvolvimento/construção de um protótipo (que será denominado de protótipo II) que permitiu a obtenção/visualização de medidas fisiológicas em tempo real. Será ainda apresentado o código que permite a obtenção dos dados fisiológicos e por último serão expostos os diferentes softwares de visualização em tempo real compatíveis com o fluxo de dados de saída, apresentando as suas interfaces, bem como as suas diversas configurações de visualização.

6.1 Hardware

Para se desenvolver um protótipo electrónico é muito importante a escolha do hardware, isto é, o conhecimento das suas caraterísticas elétricas, protocolos de comunicação e compatibilidade entre diversos dispositivos e periféricos. O protótipo II foi baseado no projeto *OpenSource* desenvolvido pelo grupo *openBCI*, o qual vêm trabalhando desde algum tempo na área da instrumentação de reabilitação médica e da realidade virtual no desenvolvimento de jogos. Sendo assim neste capítulo irá seguir-se a mesma abordagem desse grupo, para o desenvolvimento do protótipo II, uma vez que esse grupo também utilizou o dispositivo AFE ADS1299.

6.1.1 Escolha do Microcontrolador

O microcontrolador utilizado no dispositivo desenvolvido pelo grupo *openBCI* foi o *ATMega328*. Atendendo a este facto foi decidido verificar alguns requisitos, e a sua compatibilidade com a placa ADS1299, os quais são apresentados de seguida:

- ✓ Permitia a interligação com outros dispositivos por SPI;
- ✓ Tinha capacidade de processamento suficiente para lidar com um grande número de dados e com diferentes tipos de filtragens em tempo real;
- ✓ Baixo consumo de energia;
- ✓ Baixo custo;
- ✓ Fácil de programar através de uma linguagem semelhante ao C/C++, permitindo assim a utilização de diversas bibliotecas disponíveis gratuitamente;
- ✓ Permitia a integração de periféricos, nomeadamente de um módulo de comunicação sem fios (*Bluetooth*).

De referir que atualmente existem no mercado uma vasta gama de microcontroladores que poderiam satisfazer os requisitos apresentados anteriormente, entre os quais: a família de microcontroladores compatíveis com a família Arduino, os oferecidos pela *ATmel* baseados no núcleo *AVR UC3,* os *ARM Cortex-M4* e o *MSP430* da TI. No entanto o microcontrolador *ATMega328,* que se encontra presente no Arduino UNO (Figura 6.1), satisfazia todos os requisitos mencionados anteriormente, e foi o escolhido pela sua disponibilidade e facilidade

de utilização. Refira-se que o Arduino UNO tinha uma capacidade de processamento suficiente para processar os dados de saída do AFE ADS1299 em tempo real (16 MHz), e o seu custo era de aproximadamente 15 €.

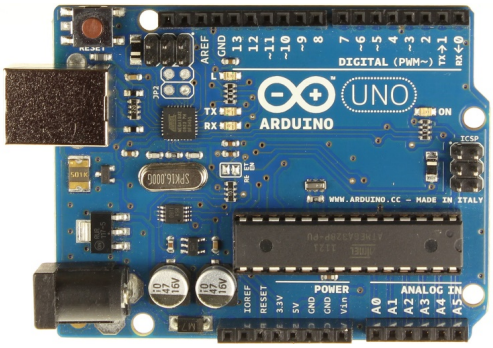


Figura 6.1 – Dispositivo Arduino UNO [39].

6.1.2 Escolha do Dispositivo de Comunicação sem Fios

Para permitir a mobilidade do protótipo desenvolvido, evitando que existisse alguma ligação física entre o protótipo e o computador, foi adicionado ao sistema um dispositivo de comunicação sem fios, nomeadamente o dispositivo *Bluetooth HC-05*, o qual pode ser visualizado na Figura 6.2.

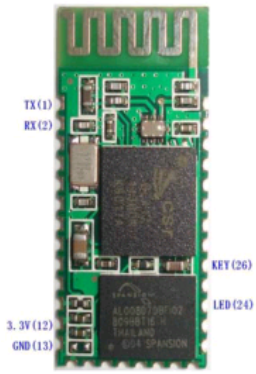


Figura 6.2 – Dispositivo *Bluetooth* HC-05 [40].

Foi escolhido este dispositivo por ser muito simples a sua ligação com outros dispositivos. É alimentado com 3,3 V e tem um consumo relativamente baixo, na ordem dos 30 mA. O protocolo de comunicação deste dispositivo utiliza dois canais, um para o envio de dados (TX) e outro para receber dados (RX) de outros dispositivos, nomeadamente do Arduino UNO. O esquema de ligação entre o HC-05 e o Arduino UNO é apresentado na Figura 6.3. O código que permite a ligação entre o Arduino UNO e o dispositivo *Bluetooth* é apresentado no Anexo E.

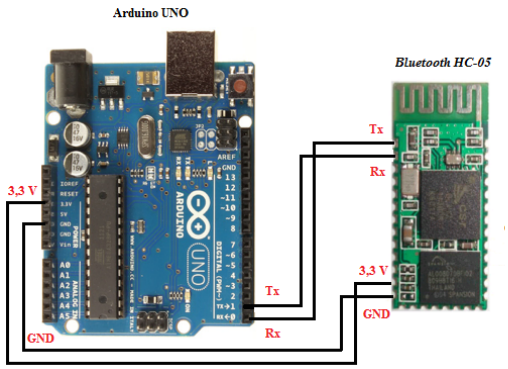


Figura 6.3 – Esquema de ligação do dispositivo *Bluetooth* HC-05 com o Arduino UNO.

A introdução deste dispositivo teve ainda a função de isolar o protótipo II de qualquer corrente elétrica nos terminais de entrada dos amplificadores que podem-se resultar num choque elétrico para o utilizador.

6.2 Alimentação

Esta secção visa explicar as opções tomadas para alimentar os diversos dispositivos que conformam o protótipo II, nomeadamente a placa de desenvolvimento AFE ADS1299, o Arduino UNO e o módulo *Bluetooth*.

Na secção 4.2.1 do capítulo 4 foi referido que o dispositivo ADS1299, é alimentado por uma tensão de 5 V e outra de 3,3 V, sendo que os pinos de comunicação SPI funcionam a 3,3 V, tal como os do módulo *Bluetooth*.

Por outro lado, o microcontrolador ATMega328 embutido no Arduino UNO pode funcionar a 3,3 V ou 5 V, sendo que no módulo UNO opera a 5 V, pelo que os seus pinos digitais funcionam a essa tensão. Esse facto é importante quando se pretende conectar este dispositivo com o ADS1299 via SPI, pois os pinos de comunicação SPI do Arduino UNO não podem ser ligados aos pinos de comunicação SPI do ADS1299 com uma tensão de 5 V, que danificaria o dispositivo. Para se resolver este problema é possível utilizar divisores de tensão, para diminuir a tensão de 5 V para 3,3 V. Apesar dessa solução ser fácil de implementar, introduz imenso ruído no sistema, devido ao factor resistivo. Atendendo ao facto anterior optou-se por modificar o módulo do Arduino UNO, fazendo-o operar a uma tensão de 3,3 V. As alterações feitas ao Arduino UNO são apresentadas no Anexo F.

6.3 Comunicação SPI

Tendo resolvido as questões de alimentação, foi possível ligar o ADS1299 ao Arduino UNO. Na secção 4.3.1 do capítulo 4 foi apresentado o esquema de ligação entre estes dois módulos, sendo o *Master* o Arduino UNO e o *Slave* o ADS1299. Os pinos de comunicação SPI do ADS1299 encontram-se no conector J3 da placa. Em relação aos pinos SPI do Arduino, estes podem ser quaisquer, mas desde que sejam pinos digitais. A correspondência dos pinos SPI de ambos dispositivos pode ser visualizada na Tabela 6.1.

Tabela 6.1 – Correspondência dos pinos de comunicação SPI do ADS1299 e do Arduino UNO [39].

Sinal	Pino do J3 do ADS1299	Pino Digital do Arduino
CS	13	10
SCLK	17	13
MISO	9	11
MOSI	7	12

6.4 Construção do Protótipo II

A construção do protótipo II foi realizada em várias fases. A primeira fase consistiu em conectar o ADS1299 ao Arduino através de cabos. A segunda fase passou pela elaboração de uma *Printed Circuit Board* (PCB) para interligar todos os dispositivos num único módulo, e a terceira e última fase consistiu em colocar o protótipo construído dentro de uma caixa, para facilitar o seu manuseamento. Nas subsecções seguintes são descritas estas fases de construção.

6.4.1 Primeira Fase

Esta primeira fase consistiu na montagem do protótipo, ou seja, em interligar o AFE ADS1299 ao Arduino UNO (Figura 6.4). Nesta fase inicial não se conectou o módulo *Bluetooth*. O Arduino foi alimentado diretamente com 3,3 V fornecidos por uma fonte de tensão de laboratório, que também foi usada para fornecer os 5 V de alimentação do ADS1299. A entrada de 3,3 V do ADS1299 foi retirada diretamente do Arduino. Nesta fase verificou-se o correto funcionamento da conexão entre estes dois dispositivos.

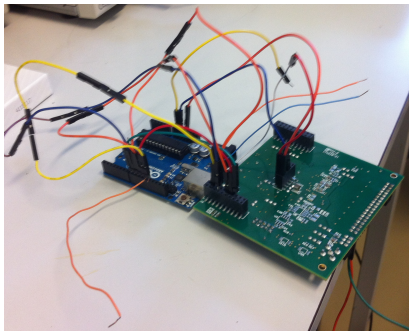


Figura 6.4 – Ligação da placa de desenvolvimento do AFE 1299 ao módulo Arduino UNO.

6.4.2 Segunda Fase

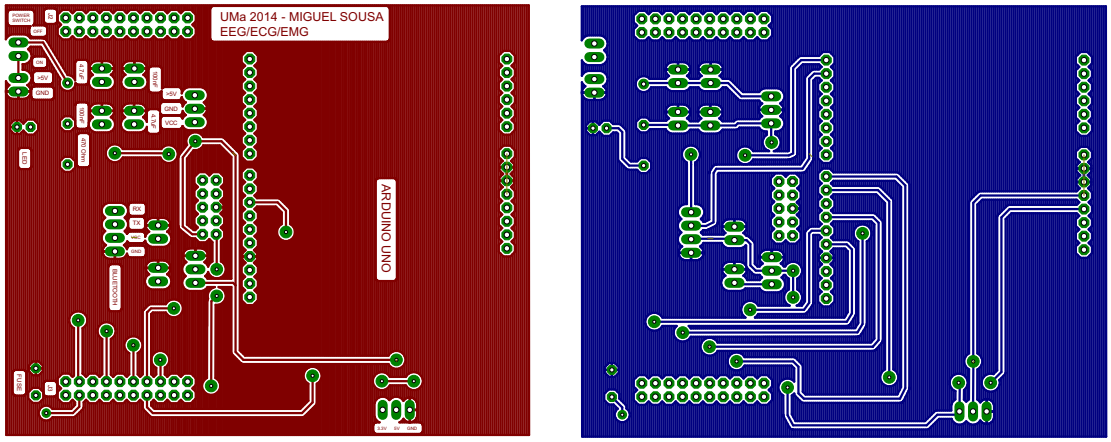
Para facilitar o manuseamento e permitir uma maior mobilidade do Protótipo, foi decidido elaborar uma placa de circuito impresso (PCB), para interligar o ADS1299, o Arduino e o módulo *Bluetooth*. O software utilizado para a criação da PCB foi o *EAGLE 6.0.1* da empresa *CadSoft*. A integração dos dispositivos numa PCB é um processo complexo, visto

que é necessário garantir um determinado espaço para os dispositivos, para as suas ligações e ainda estudar qual a melhor posição em que estes devem ser colocados.

Na elaboração da PCB para além dos módulos referidos anteriormente foram adicionados dois reguladores de tensão, um de 5 V e outro de 3,3 V, respetivamente, com o intuito de substituir a fonte de tensão que alimentava os módulos por um *kit* de 8 pilhas recargáveis, o qual pode ser visualizado na Figura 6.5, garantindo uma maior segurança e portabilidade ao utilizador. O conjunto de 8 pilhas fornecia 9,6 V, sendo esta tensão reduzida para 5 V pelo regulador de tensão de 5 V (que alimentava o ADS1299) e posteriormente a saída desse regulador era novamente regulado para 3,3 V (que alimentavam o ADS1299 e o Arduino).

Figura 6.5 – Kit de 8 baterias.

Na Figura 6.6-a, pode-se visualizar o esquema da PCB vista de topo e na Figura 6.6-b pode-se visualizar o esquema da PCB vista de baixo.

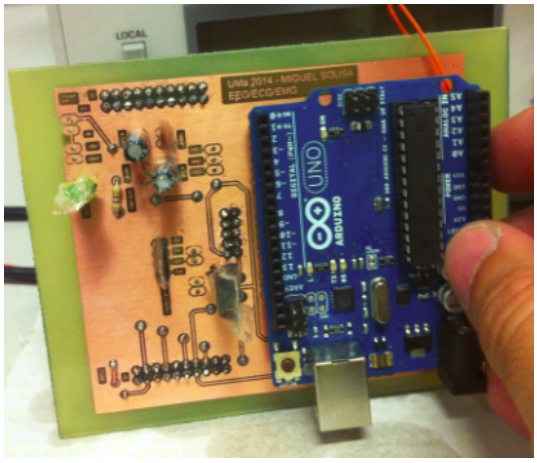


(a) (b)

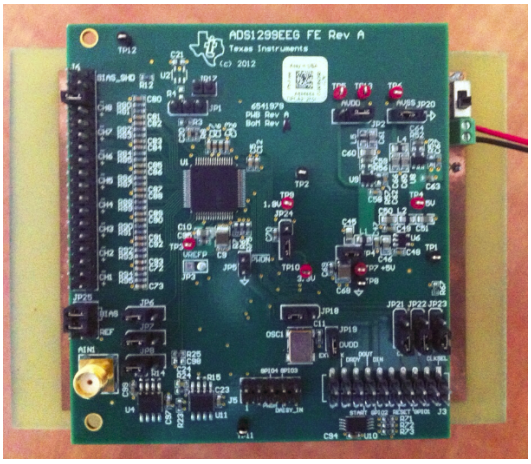
Figura 6.6 – PCB Desenhada: (a) PCB vista do Topo ; e (b) PCB vista de baixo.

Posteriormente procedeu-se ao fabrico da placa, o que passou essencialmente por cinco passos: Impressão em papel de acetato, sensibilização por UV, revelação das pistas e remoção do cobre, preparação da placa para soldar e por último a montagem dos componentes.

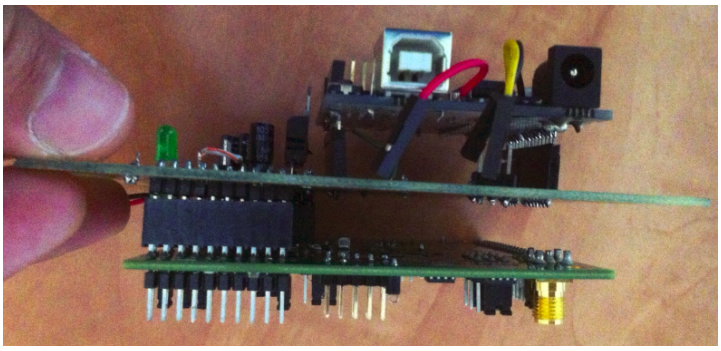
Uma vez concluída a placa realizaram-se diversos testes. O primeiro consistiu na verificação de possíveis curtos-circuitos existentes na PCB. De seguida alimentou-se a placa com o *kit* de baterias e verificou-se se todas as tensões da placa correspondiam as esperadas. Realizados esses testes foram colocados na PCB o Arduino UNO (Figura 6.7-a) e o ADS1299 AFE (Figura 6.7-b). Na Figura 6.7-c, encontra-se o módulo desenvolvido visto de um dos lados.



(a)



(b)

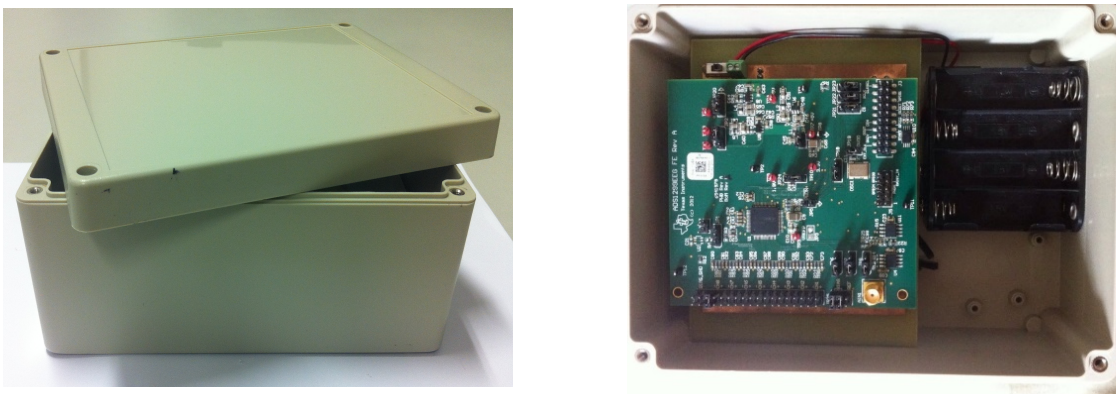


(c)

Figura 6.7 – PCB construída: (a) Colocação do módulo Arduino UNO na PCB *Bottom*; (b) Colocação do AFE ADS1299 na PCB *Top;* e (c) PCB vista de um dos lados.

6.4.3 Terceira Fase

Esta fase consistiu em colocar o protótipo II dentro de uma caixa de plástico, a qual pode ser visualizada na Figura 6.8-a. O espaço da caixa era adequado à colocação do protótipo II e do seu *kit* de pilhas, como mostra a Figura 6.8-b.

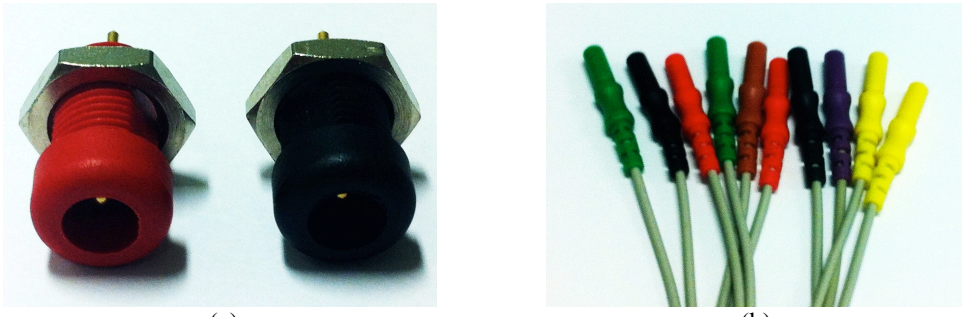


(a) (b)

Figura 6.8 – Protótipo II: (a) Caixa de plástico para colocação do protótipo II; e (b) Protótipo II e o *kit* de pilhas inserido dentro da caixa.

O *kit* de pilhas foi colocado de modo a que este fosse fácil de retirar da caixa, para se poderem recarregar as pilhas. Observando a Figura 6.8-b, pode-se ver que o ADS1299 foi colocado para cima, para se poder aceder aos seus diversos módulos com facilidade. Apesar do módulo *Bluetooth* não ser visível na Figura 6.8-b, este encontra-se por baixo da PCB desenvolvida juntamente com o Arduino. Foi realizado um teste de conectividade da ligação sem fios e esta foi conseguida com sucesso.

O passo seguinte foi disponibilizar o acesso aos diversos módulos do ADS1299, nomeadamente os 8 canais, os sinais de referência e o pino de massa (*ground*), sem ter que se aceder ao interior da caixa. Para isso foram feitos buracos na caixa e colocados conectores do tipo *DIN* 42 802 fêmea (Figura 6.9-a), pois esses permitem ligar os elétrodos que tem uma terminação com conectores do tipo *DIN* 42 802 macho (Figura 6.9-b).



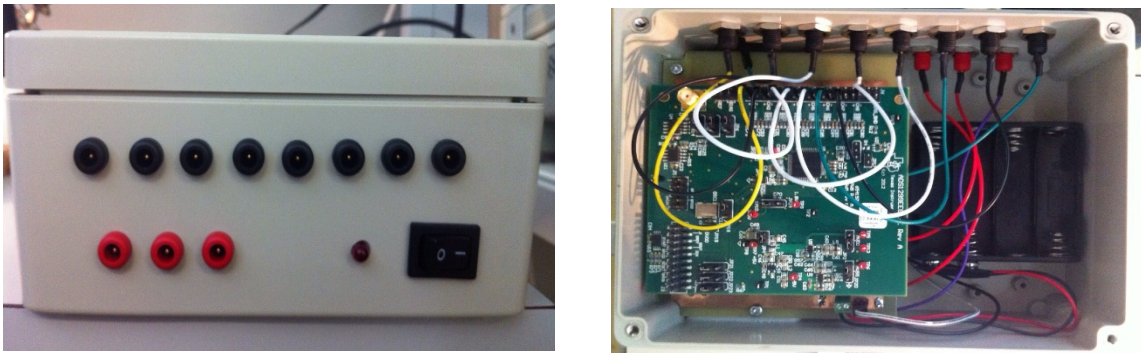
(a) (b)

Figura 6.9 – Conectores do tipo *DIN 42 804*: (a) Conectores *DIN* Fêmeas; e (b) Terminação dos elétrodos com conectores *DIN* Machos.

A ligação entre os pinos de acesso do ADS1299 e os conectores *DIN* 42 802 foi feita com cabos *PIN Headers* fêmea-fêmea (Figura 6.10), aos quais foi retirada numa das suas extremidades a sua terminação fêmea, para se poder soldar o conector *DIN* 42 802. De referir que foi adicionada manga térmica nessa soldadura para garantir uma melhor segurança.

Figura 6.10 – Cabo PIN *Header* fêmea-fêmea, com o conector *DIN* 42 802 soldado numa das duas extremidades.

Os conectores foram adicionados à caixa de plástico do protótipo II. Ainda no exterior da caixa foi colocado um interruptor que permitia ligar/desligar o protótipo II e um LED vermelho indicando o estado desse interruptor. Na Figura 6.11-a, pode-se visualizar a colocação dos conectores *DIN* 42 802, interruptor e led no exterior da caixa, e na Figura 6.11-b pode-se ver a ligação entre os conectores *DIN* 42 802 e o ADS1299.



(a) (b)

Figura 6.11 – Caixa de plástico do protótipo II: (a) Conectores *DIN* 42 802, interruptor e led colocados no exterior da caixa; e (b) Conexão dos conectores *DIN*42 802 ao ADS1299.

Posteriormente fez-se um teste ao protótipo para verificar o seu correto funcionamento e verificou-se que o Arduino não poderia ser programado com o módulo *Bluetooth* ligado a ele. Para resolver este problema foi adicionado um outro interruptor que permitia ligar/desligar o dispositivo *Bluetooth*. Esse interruptor tinha duas funcionalidades diferentes, a já referida anteriormente, e a segunda que permitia ao utilizador escolher entre duas formas de aquisição de medidas fisiológicas: ou por cabo (via USB) ou por comunicação sem fios (via *Bluetooth*).

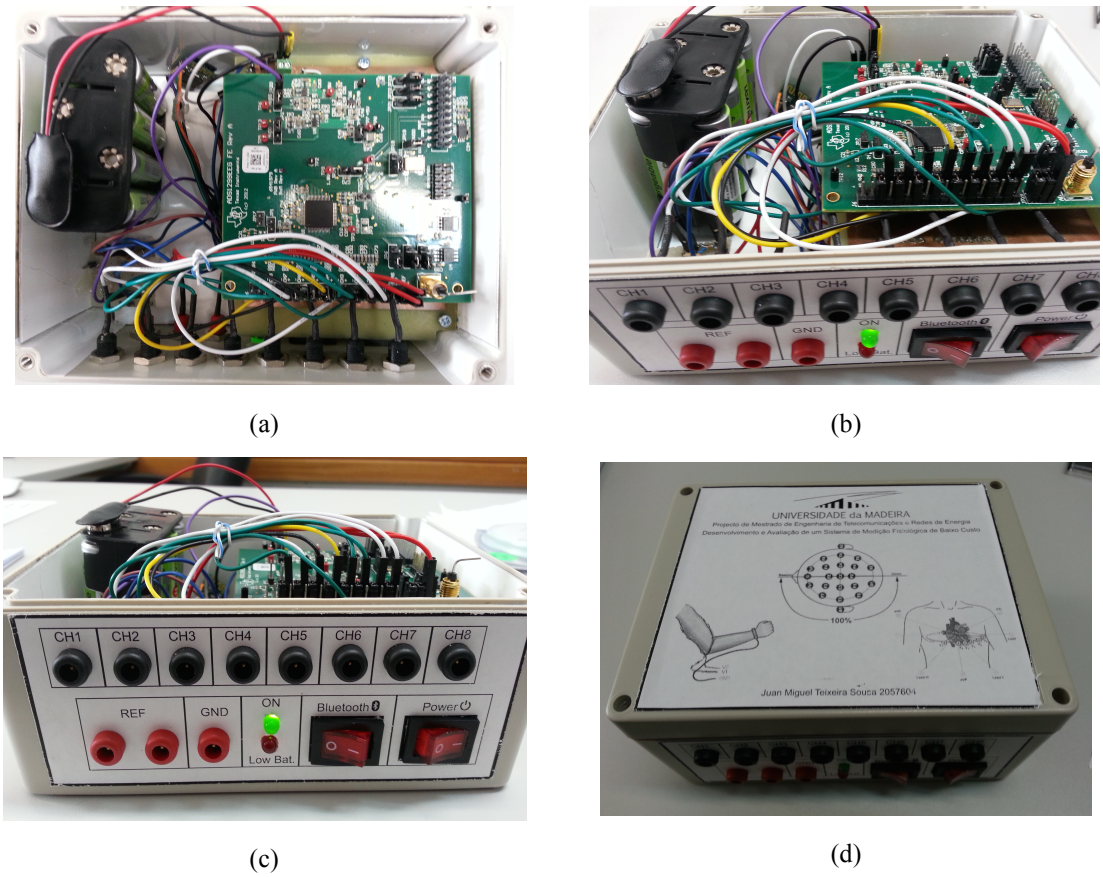
Com o intuito de distinguir a função de cada um dos conectores no exterior da caixa, foi colocado um autocolante na caixa com a descrição de cada um desses conectores, como se pode visualizar na Figura 6.12. De referir que os conectores *DIN* foram escolhidos de cor preta para cada um dos 8 canais e vermelha para as referências e o *Ground*.

Figura 6.12 – Caixa do Protótipo II.

Após se realizarem diversos testes ao protótipo II, entre os quais, testes de conectividade sem fios a diferentes distâncias do PC a que se encontrava ligado, verificou-se que o mesmo não estava a receber os dados corretamente ou que se perdia a ligação sem fios com o PC para uma distância relativamente curta e em linha de vista, de aproximadamente 3 metros. Com o intuito de resolver este problema colocou-se o módulo *Bluetooth* perto do *kit* de baterias, verificando-se assim que o módulo conseguia transmitir os dados com sucesso, sem se perder a ligação para uma distância superior a mencionada anteriormente.

Ainda no decorrer dos testes anteriores, foi decidido colocar um outro led no exterior da caixa, com objetivo de avisar o utilizador do estado das baterias, e assim tentar não danificar as mesmas, ou seja, não permitindo que o seu nível de tensão baixe de um determinado nível e, possam ser recarregadas novamente sem problemas. Assim o led vermelho que já se encontrava no exterior da caixa passou a ser o indicador de nível baixo de baterias (*low battery*) e o led adicionado, de cor verde, indicava que o nível das baterias se encontra adequado para a obtenção de medidas fisiológicas. O circuito que permitia medir o nível de tensão das baterias consistiu num divisor resistivo, com um amplificador que servia de comparador, sendo este circuito simples e fácil de se colocar dentro da caixa. Uma outra opção seria utilizar uma porta analógica do Arduino UNO, e programar o mesmo para medir o nível de tensão. Como foi referido anteriormente o *kit* de baterias tem uma tensão de aproximadamente de 9,6 V, tendo o circuito de medição do nível de tensão sido dimensionado para este valor, ficando o led vermelho ativo para uma tensão abaixo dos 7,5 V, indicando que as baterias já não possuem um nível de tensão adequado para se trabalhar.

Com todas estas alterações o visual tanto interno como externo da caixa foi alterado, podendo o resultado final ser visualizado na Figura 6.13, onde se pode observar que os fios que interligam os conectores *DIN* 48 802 ao ADS1299 foram acomodados devidamente para um melhor manuseamento do protótipo II. De salientar que também foram alteradas as funções de dois conectores *DIN*, nomeadamente os dos sinais de polarização (indicados como BIAS, no exterior da caixa), visto os mesmos não serem utilizados neste projeto, passando um desses conectores a ser um segundo sinal de referência e o outro com a função de massa (*ground*).



(a) (b)

(c) (d)

Figura 6.13 – Visual final do Protótipo II: (a) visto de cima; (b) visto de lado; (c) parte de frente; e (d) caixa com a tampa.

6.5 Custo do Protótipo II

Uma vez construído o protótipo II avaliou-se o custo total do mesmo. Na Tabela 6.2, são apresentados os dispositivos e componentes, com o respetivo preço. Os preços datam de 01 de setembro de 2015.

De mencionar que nesta avaliação do custo do protótipo II não foram incluídos os elétrodos EGG, ECG ou EMG, os cabos de ligação ao protótipo II, nem o capacete EEG nem o gel de condução eletrólito.

Como se pode visualizar na Tabela 6.2, o preço total do protótipo II é de aproximadamente 300 €, sendo os dispositivos mais caros essenciais para o funcionamento do protótipo desenvolvido.

Tabela 6.2 – Descrição e preço dos dispositivos e componentes do Protótipo II.

Qt.	Descrição do Produto	Preço (€)
1	ADS1299 + DSP	175
1	Arduino UNO	22
1	*Bluetooth* HC-05	3
1	Regulador de Tensão 5 V	1,5
1	Regulador de Tensão 3,3 V	2
2	Condensador 4,7 µF	1
2	Condensador 100 µF	1
2	LED	0,40
2	Interruptor	0,5
1	PCB dupla fase	5
11	Conectores *DIN* 42 804 Fêmea	39,90
10	Cabos PIN *Headers*	1,50
1	Kit para pilhas + 8 pilhas recarregáveis	21,50
1	1 metro de manga térmica	1,50
1	Caixa elétrica de plástico	20
	TOTAL	**295,50**

6.6 Código e Programação do Protótipo II

Concluída a construção do protótipo II deu-se início à programação do mesmo. Tal como na escolha do microcontrolador, neste passo decidiu-se utilizar o código desenvolvido em linguagem C++ pelo grupo *openBCI*. A programação do Arduino foi efetuada utilizando o software Arduino IDE (Figura 6.14), o qual permite compilar e visualizar o resultado do código compilado. Inicialmente foi necessário escolher o modelo do Arduino que se pretendia programar e a sua porta de comunicação, o que foi feito através do menu *Tools –> Boards* e *Tools –> Serial Port*, respetivamente.

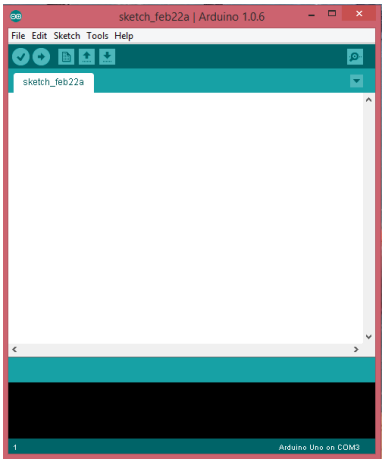


Figura 6.14 – Janela inicial do software Arduino IDE.

Quando se trabalha com código desenvolvido por outras pessoas e/ou grupos é necessário ter em consideração vários aspetos que vão desde a sua estruturação e importação dos ficheiros, estudo do conteúdo e funcionalidade de cada um dos seus ficheiros, possibilitando assim adaptar esse código ao nosso projeto, alterando ou até removendo o que

realmente não será preciso. Nas seguintes subsecções serão apresentadas as alterações realizadas no código.

6.6.1 Estrutura do Código

Nesta secção é apresentada a estrutura do código desenvolvido pelo grupo *OpenBCI* e ainda como devem ser importados os ficheiros necessários para o seu correto funcionamento. O código encontra-se subdividido em três pastas, as quais são apresentadas de seguida:

Pasta ADS1299: Esta pasta é composta por 5 ficheiros em linguagem C++. Esta pasta deve ser importada para a pasta *Library* do diretório do software Arduino IDE instalado previamente no PC. Os ficheiros são os seguintes: ADS1299Manager.cpp, ADS1299Manager.h, ADS1299.cpp, ADS1299.h e o último Definitions.h. De salientar que estes ficheiros tem a configuração dos registos que conformam o dispositivo ADS1299, permitindo a obtenção de dados, sendo estes abordados mais a frente.

Pasta Biquad: Composta por 4 ficheiros, desenvolvidos também em linguagem C++, e que são os seguintes: Biquad.cpp, Biquad.h, Biquad_multiChan.cpp e o Biquad_multiChan.h. Estes ficheiros foram desenvolvidos por diversos grupos *OpenSource* que trabalham na área das medições fisiológicas com diversos dispositivos existentes no mercado. De salientar que estes ficheiros não foram criados de propósito para o dispositivo desenvolvido pelo grupo *openBCI*, mas sim alterados consoante as suas necessidades. Os ficheiros contém diversos tipos de filtragens, comummente utilizadas nesta área, podendo serem adaptados com facilidade, dependendo do tipo de projeto em causa. Exemplos dessas filtragens são: filtragem das componentes das redes elétricas, 50 e/ou 60 Hz (*notch filter*), filtragem de altas frequências (*low pass filter*), nomeadamente acima dos 200 Hz, e filtragens de baixa frequência (*high pass filter*), entre os 0 e os 3 Hz. A pasta Biquad também deve ser importada para a pasta *Library* do diretório do software Arduino IDE.

Pasta principal (main): Corresponde a pasta do ficheiro principal do código, sendo denominado de StreamRawData.ino. Este contém as bibliotecas, funções principais e a interface que é apresentada ao utilizador, sendo que será abordado detalhadamente mais à frente, apresentando-se as diversas escolhas/alterações realizadas para o correto funcionamento do protótipo II.

6.6.2 Apresentação do Código

Nesta secção será apresentado o código que foi programado no protótipo II, para a obtenção dos sinais fisiológicos pretendidos neste projeto.

Foi referido no Capítulo 4 que o dispositivo ADS1299 é programado através do envio de *opcodes* por meio da comunicação SPI. No ficheiro principal do código encontram-se todas as variáveis globais, definição e inclusão de bibliotecas, o ciclo principal, e as diversas funções que definem o estado dos registos internos do ADS1299. Com o objetivo de se

perceber melhor o funcionamento do código serão apresentadas as partes primordiais do mesmo:

Definição do Número de Canais: O grupo *openBCI* definiu uma variável para determinar o número de canais disponíveis no ADS1299, que se sabe que é de 8. Esta variável foi definida da seguinte maneira:

```
//number of channels on a single OpenBCI board
#define N_CHANNELS_PER_OPENBCI (8)
```

Esta variável foi criada com o intuito de poder ser expandido o número de canais, quando se utilizam dois ou mais dispositivos ligados em cascata, o que não foi o caso deste projeto.

De modo a se poderem incluir os ficheiros que foram importados para as livrarias do Arduino IDE foi utilizado o seguinte código:

```
//for using a single OpenBCI board
#include <ADS1299Manager.h>
//Uses SPI bus and pins to say data is ready.
ADS1299Manager ADSManager;
//how many channels are available in hardware
#define MAX_N_CHANNELS (N_CHANNELS_PER_OPENBCI)
//how many channels are available in hardware...use this for daisy-
chained board
//#define MAX_N_CHANNELS (2*N_CHANNELS_PER_OPENBCI)
int nActiveChannels = MAX_N_CHANNELS;
```

O código apresentado anteriormente tinha a função de verificar quando se estava a utilizar um único dispositivo ou vários ligados em cascata.

Definição do Tipo de Entrada do ADS1299: O ADS1299 possui várias opções de entrada, que foram apresentadas no capitulo 4.

Para facilitar foram colocadas unicamente as opções que iriam ser necessárias neste projeto, nomeadamente: entrada externa de elétrodos e/ou gerador de sinais (ADSINPUT_NORMAL), canais curto-circuitados (ADSINPUT_SHORTED) e geração/calibração de ondas geradas internamente pelo dispositivo (ADSINPUT_TESTSIG). Em relação ao ganho deste tipo de entradas foi decidido definir um ganho de 24, devido às baixas amplitudes dos sinais fisiológicos. O código referente a esta definição é o seguinte:

```
//how much gain do I want
byte gainCode = ADS_GAIN24;
//here's the normal way to setup the channels
byte inputType = ADSINPUT_NORMAL;
//here's another way to setup the channels
byte inputType = ADSINPUT_SHORTED;
//here's a third way to setup the channels
byte inputType = ADSINPUT_TESTSIG;
```

Formato dos Dados: Os formatos dos dados de saída possíveis são os seguintes: dados em texto, em binário e em hexadecimal. Neste projeto foi utilizado o formato de dados em texto, sendo que para esse fim foram criados alguns comandos de envio por SPI, para facilitar a tarefa ao utilizador, os quais podem ser observados na Tabela 6.3.

Tabela 6.3 – Comandos SPI que podem ser enviados pelo utilizador do Protótipo II.

Comando SPI	Função
X	Inicializa a conversão de dados em texto
B	Inicializa a conversão de dados em binário
H	Inicializa a conversão de dados em hexadecimal
S	Para a conversão de dados
?	Mostra os valores de todos os registos a qualquer instante
1 / Q	Ativa o canal 1 / Desativa o canal 1
2 / W	Ativa o canal 2 / Desativa o canal 2
3 / E	Ativa o canal 3 / Desativa o canal 3
4 / R	Ativa o canal 4 / Desativa o canal 4
5 / T	Ativa o canal 5 / Desativa o canal 5
6 / Y	Ativa o canal 6 / Desativa o canal 6
7 / U	Ativa o canal 7 / Desativa o canal 7
8 / I	Ativa o canal 8 / Desativa o canal 8
F	Ativa a filtragem
f	Desativa a filtragem

De referir que existem outros comandos definidos no código que não foram colocados na Tabela 6.3, porque não foram utilizados neste projeto.

Filtragem: A filtragem na área das medições fisiológicas é muito importante pelo que o grupo *openBCI* não deixou para trás este facto e utilizou diversas bibliotecas criadas por vários grupos *OpenSource*. O código desenvolvido para incluir essas bibliotecas foi o seguinte:

```
    //Design filters
    #include <Biquad_multiChan.h>     //modified from this source code: http://www.earlevel.com/main/2012/11/26/biquad-c-source-code/
    #define SAMPLE_RATE_HZ (250.0)  //default setting for OpenBCI
    #define FILTER_Q (0.5)               //critically damped is 0.707 (Butterworth)
    #define FILTER_PEAK_GAIN_DB (0.0) //we don't want any gain in the passband
    #define HP_CUTOFF_HZ (0.5)  //set the desired cutoff for the highpass filter
    Biquad_multiChan stopDC_filter(MAX_N_CHANNELS,bq_type_highpass,HP_CUTOFF_HZ / SAMPLE_RATE_HZ, FILTER_Q, FILTER_PEAK_GAIN_DB); //one for each channel because the object maintains the filter states
    //Biquad_multiChan  stopDC_filter(MAX_N_CHANNELS,bq_type_bandpass,10.0 / SAMPLE_RATE_HZ, 6.0, FILTER_PEAK_GAIN_DB); //one for each channel because the object maintains the filter states
    #define NOTCH_FREQ_HZ (60.0)
    #define NOTCH_Q (4.0)              //pretty sharp notch
    #define NOTCH_PEAK_GAIN_DB (0.0)   //doesn't matter for this filter type
    Biquad_multiChan notch_filter1(MAX_N_CHANNELS,bq_type_notch,NOTCH_FREQ_HZ / SAMPLE_RATE_HZ, NOTCH_Q, NOTCH_PEAK_GAIN_DB); //one for each channel because the object maintains the filter states
```

```
    Biquad_multiChan
notch_filter2(MAX_N_CHANNELS,bq_type_notch,NOTCH_FREQ_HZ / SAMPLE_RATE_HZ,
NOTCH_Q, NOTCH_PEAK_GAIN_DB); //one for each channel because the object
maintains the filter states
    boolean useFilters = false;  //enable or disable as you'd like...turn
off if you're daisy chaining!
```

Atendendo ao código anterior teve-se em conta que a rede elétrica europeia funciona a 50 Hz e não a 60 Hz, pelo que o factor de frequência foi alterado em todos os filtros passa banda e, também foram recalculados o seus coeficientes, usando a ferramenta *MATLab*.

Comunicação Série e Taxa de Transmissão dos dados: Devido à capacidade de processamento do Arduino UNO (16 MHz) e do módulo *Bluetooth*, foi decidido que o protótipo II operaria a uma taxa de transmissão de dados de 250 SPS, por conseguinte a comunicação série entre os equipamentos foi feita a 115,2 kbits/s, sendo esta suficiente para transmitir os dados dos 8 canais em simultâneo sem erro.

Leitura/Escrita dos registos do ADS1299: O método de leitura/escrita dos registos internos do dispositivo também foi apresentado no capítulo 4, verificando-se que os mesmos contém algumas restrições. Analisando o código chegou-se à conclusão que as funções que permitem a leitura/escrita foram definidas tendo em consideração todas essas restrições e a sequência de envio dos *opcodes*, pelo que estas funções não foram alteradas, mantendo-se assim as desenvolvidas pelo grupo *openBCI* neste projeto.

Configuração dos Registos do ADS1299: A configuração dos registos internos do dispositivo ADS1299 foi de elevada importância e será apresentada de seguida.

O código apresenta uma configuração dos registos que permite a obtenção das ondas geradas internamente pelo dispositivo ADS1299. Por essa razão, no capítulo 4, foram descritos cada um dos registos internos desse dispositivo e as suas diversas configurações, com o objetivo de se saber a função de cada um deles, sendo assim mais fácil a sua modificação (Tabela 6.4). De referir que tais registos foram modificados de modo a que o protótipo II fosse capaz de ler os sinais dos elétrodos (EEG, ECG e EMG), com uma referência comum entre eles gerada internamente e positiva.

Tabela 6.4 – Registos alterados no software.

Registo	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CONFIG1	1	0	0	1	0	1	1	0
CONFIG2	1	1	0	0	0	0	0	1
CONFIG3	1	1	1	0	0	0	0	0
CHnSET	0	1	1	0	0	0	0	0
GPIO	0	0	0	0	1	1	1	1
MISC1	0	0	1	0	0	0	0	0

Os restantes registos que não foram modificados, e ficaram com o valor que vem definido por defeito (*default*) e que podem ser visualizados no Anexo A.

6.6.3 Modo de funcionamento do dispositivo ADS1299

Para além da configuração dos registos teve-se em conta que o dispositivo ADS1299 permite que os canais funcionem em modo unipolar ou em modo diferencial. Para este projeto decidiu-se utilizar os dois modos, sendo que para o modo unipolar foram utilizadas as entradas positivas de cada canal (Anexo G), ficando as entradas negativas curto-circuitadas internamente no dispositivo e, para o modo diferencial foram utilizadas ambas as entradas (Anexo H), a positiva e a negativa de cada canal.

Em relação à referência comum, esta envolveu a alteração do *jumper* JP8 no dispositivo ADS1299 (para além da modificação dos respetivos registos referidos na secção anterior), sendo colocado nos pinos 1 e 2, ativando assim a referência comum gerada internamente pelo dispositivo, ficando a ligação física disponível nos pinos 5 e 6 do JP25.

A ligação do elétrodo de massa, podia ser feita em vários pontos do dispositivo ADS1299, no entanto foi escolhido o pino 1, do *jumper* JP20.

6.6.4 Programação do Protótipo II

Após o estudo e respetivas modificações do código, procedeu-se à programação do protótipo II, através do software Arduino IDE apresentado anteriormente. Todo o código envolvido nesta fase de programação encontra-se no Anexo I, devidamente comentado para uma melhor percepção. Depois de programado o protótipo II realizou-se o primeiro teste para verificar se o resultado das alterações realizadas no código era o pretendido, sendo verificado o estado de cada um dos registos internos do ADS1299, os quais foram verificados com sucesso.

6.7 Visualização dos Dados do Protótipo II

Quando se está a trabalhar na área das medidas fisiológicas é de elevada importância poder visualizar os traços das mesmas em tempo real, tanto no domínio do tempo como no domínio das frequências, e ainda que esses dados possam ser guardados para posterior análise. Nas próximas subsecções serão apresentadas as diversas formas de visualização dos dados do protótipo II.

6.7.1 Visualização dos Dados no Software Arduino *IDE*

O software Arduino IDE permite visualizar o resultado do código apresentado anteriormente, através da janela *Serial Monitor* (Figura 6.15). Como os dados são transferidos por uma porta série predefinida pelo sistema operativo do PC, quaisquer programas que permitam aceder a essa porta, tais como o *Putty*, conseguem visualizar os dados de saída do protótipo II. Através da janela *Serial Monitor*, eram apresentados os valores iniciais dos registos e também era possível o envio dos comandos SPI referidos na secção anterior, tal como se pode observar na Figura 6.15.

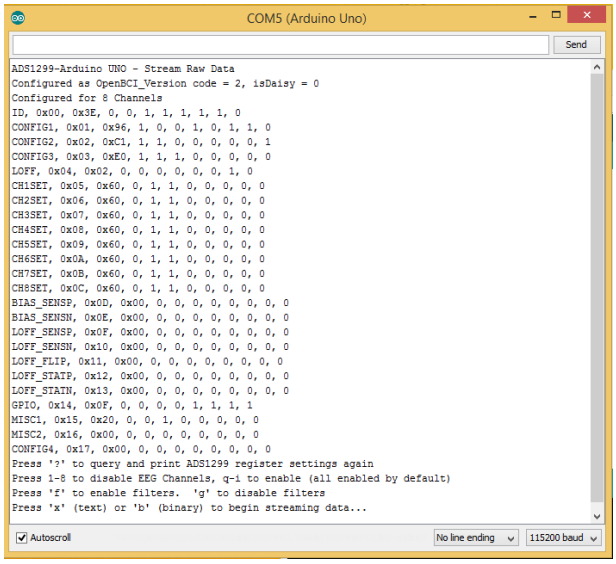


Figura 6.15 – Janela *Serial Monitor* do software Arduino IDE.

Apesar de neste software se conseguirem obter os dados de cada canal em tempo real, nesta forma de visualização não se conseguiam ver os traços dos mesmos em tempo real.

6.7.2 Software *openBCI*

De modo a permitir esse tipo de visualização o grupo *openBCI* criou um software *OpenSource* em linguagem *Java* que permite visualizar os traços dos dados de saída do ADS1299. Na Figura 6.16 pode ser visualizada a interface desse software.

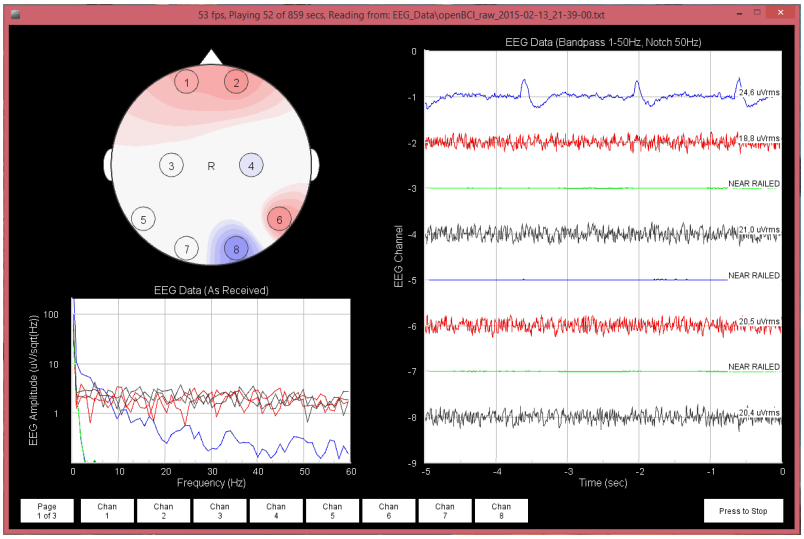


Figura 6.16 – Interface do software desenvolvido pelo grupo *OpenSource openBCI.*

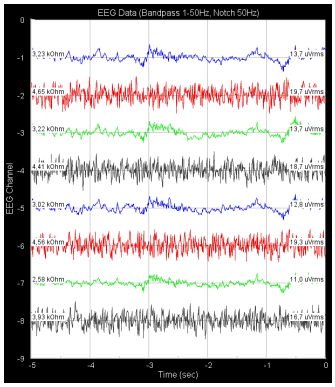
A interface gráfica que é apresentada ao utilizador encontra-se dividida em 5 partes que são apresentadas de seguida:

Menu Principal: A interface disponibiliza três menus que permitem ao utilizador escolher diversas opções de visualização das medidas fisiológicas. O primeiro menu permite ligar/desligar os 8 canais disponíveis no ADS1299. Na Figura 6.17 encontra-se representado esse menu sendo que os primeiros 4 canais se encontram ligados (cor branca) e os últimos 4 se encontram desligados (cor cinzenta), exemplificando a função deste primeiro menu.

Figura 6.17 – Menu que permite ligar/desligar os canais do ADS1299.

O segundo menu (Figura 6.18-a) permite visualizar o valor de impedância de cada um dos canais. Esses valores são apresentados no lado esquerdo do gráfico de canal, como se pode visualizar na Figura 6.18-b. De mencionar que "Imp P1" refere-se ao valor da impedância da entrada positiva do canal 1 e "Imp N1" refere-se ao valor da impedância da entrada negativa do canal 1 e, assim sucessivamente para todos os canais.

(a)



(b)

Figura 6.18 – Visualização do valor da impedância de cada canal: (a) Menu que permite ligar/desligar a visualização do valor da impedância de cada canal; e (b) Visualização do valor da impedância de cada canal.

O terceiro e último menu (Figura 6.19) permite escolher o sistema de filtragem, a escala de apresentação no domínio dos tempos e das frequências das medidas fisiológicas.

Figura 6.19 – Menu que permite escolher o sistema de filtragem, escala no domínio dos tempos e das frequências.

Gráfico no domínio dos tempos: A interface apresenta as medidas fisiológicas no domínio dos tempos traçando os seus valores em gráficos (Figura 6.20). De referir que no lado direito do gráfico de cada canal se encontra o valor RMS do valor medido.

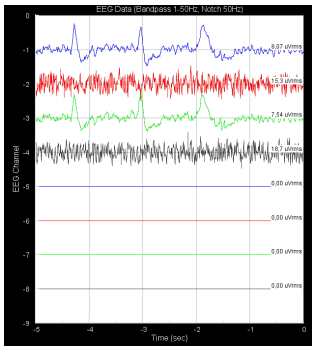


Figura 6.20 – Apresentação gráfica das medidas fisiológicas no domínio dos tempos.

Gráfico no domínio das frequências: Quando se trabalha na aquisição de sinais fisiológicos, nomeadamente do EEG, é de elevada importância a apresentação das medidas no domínio das frequências. Esta interface apresenta um gráfico nesse domínio, o qual pode ser visualizado na Figura 6.21.

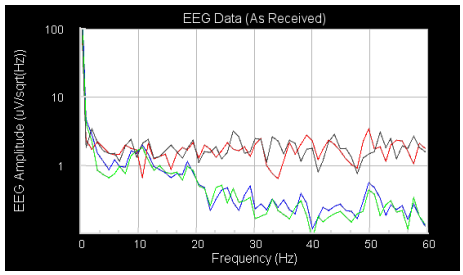


Figura 6.21 – Apresentação gráfica das medidas fisiológicas no domínio das frequências.

Posicionamento dos elétrodos: A interface também apresenta o posicionamento dos elétrodos numa imagem, como pode ser visualizado na Figura 6.22. Nessa imagem pode-se visualizar a intensidade do sinal em cada elétrodo, sendo que a cor vermelha indica grande intensidade do sinal, a cor rosa indica uma intensidade média, a cor azul indica que quase não há intensidade do sinal e finalmente a cor branca indica que o canal se encontra desligado.

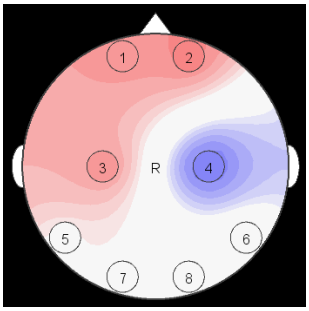


Figura 6.22 – Posicionamento dos elétrodos indicando a sua intensidade de sinal.

Dados Obtidos: Cada vez que se faz uma aquisição de dados estes são guardados num ficheiro de texto. O nome do ficheiro, bem como a sua localização podem ser visualizado na barra superior da interface. Esta opção permite exportar os dados para outras ferramentas como o *MATLab* ou até mesmo o *Excel*.

No ficheiro são guardados os valores em 10 colunas diferentes (Figura 6.23). A primeira coluna contém o número da amostra, as colunas de 2 até a 9 contém os valores das medidas

fisiológicas de cada um dos canais e a última coluna encontra-se sempre com o valor zero para indicar o fim da trama.

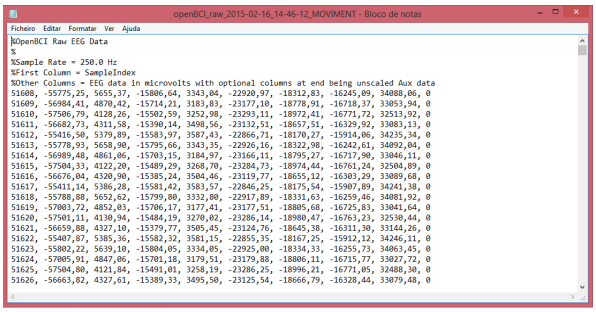


Figura 6.23 – Formato do ficheiro de texto onde são guardados os valores das medidas fisiológicas adquiridas.

6.7.3 Software *MATLab*

O software *MATLab* também poderá ser utilizado para importar os dados das medições fisiológicas realizadas com o Protótipo II. Para isso deverão ser exportados os dados guardados pelo software Arduino IDE ou pelo *openBCI* para o *MATLab* e posteriormente manipulados e traçados da forma que o utilizador precisar. Refira-se que neste projeto não se tinha o objetivo de estudar e analisar os dados fisiológicos obtidos, ou seja, o seu pós-processamento, mas mesmo assim no próximo capítulo será apresentada a importação dos ficheiros de texto gerados para o *MATLab*, servindo de apoio a futuros projetos de análise dos dados em pós-processamento.

6.8 Conclusões do Capítulo

Inicialmente neste capítulo foi apresentado o microcontrolador que fez parte do hardware no desenvolvimento do protótipo II, sendo este escolhido com base no projeto do grupo *openBCI* e após terem sido verificados alguns requisitos apresentados ao longo deste projeto. Viu-se também neste capítulo que foi adicionado um módulo de comunicação sem fios (*Bluetooth*) para permitir alguma mobilidade ao utilizador do protótipo II. Em relação à alimentação do protótipo II foi decidido utilizar baterias para o alimentar, garantindo assim segurança ao utilizador. Quando se desenvolve um protótipo electrónico é essencial determinar o seu custo, para assim poder ser comparado com outros dispositivos existentes no mercado. Verificou-se que o protótipo II apresentava um custo total de aproximadamente 300 €, o qual se encontra muito abaixo quando comparado com os dispositivos apresentados no final do capítulo 2. No entanto, só no capitulo 7 é que será analisada a sua relação custo/performance. Finalmente e talvez a parte mais importante deste capítulo, foi a apresentação do código utilizado para programar o microcontrolador do protótipo II e as suas modificações para se poderem atingir os objetivos pretendidos neste projeto. Foram ainda apresentados os diversos softwares de visualização dos dados em tempo real compatíveis com os dados gerados pelo protótipo II.

7 Resultados Obtidos

Neste capítulo serão apresentados os testes e os resultados obtidos com o protótipo II. Os testes foram divididos em 3 fases: a primeira consistiu na verificação do ruído do protótipo II, a segunda abrangeu a aquisição das ondas geradas internamente pelo dispositivo ADS1299 e a última fase consistiu na obtenção de ondas geradas por um gerador de sinais. Em relação à obtenção dos sinais fisiológicos (EEG, ECG e EMG) com o protótipo II esses resultados serão apresentados no final deste capítulo, sendo também apresentados resultados obtidos com outros dispositivos existentes no laboratório da UMa/M-ITI, nomeadamente o *g.MOBIlab+*, o *Enobio NE*, o *PLUX* e o *BITalino BioSignals*, com o intuito de comparar e avaliar a performance do protótipo II.

7.1 Medição do Ruído no Protótipo II

Para a medição do ruído no protótipo II seguiu-se o mesmo procedimento realizado com o protótipo I, apresentado no capítulo 5. Na Figura 7.1, pode-se observar o ruído obtido após a medição com o protótipo II.

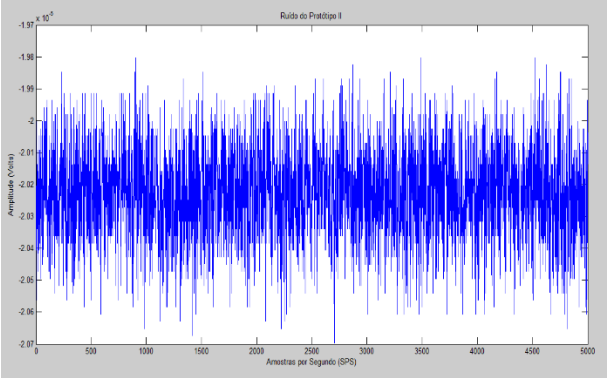


Figura 7.1 – Ruído medido com o Protótipo II.

O valor de V_{PP} medido com o protótipo II foi de 0,894 μV_{PP}, sendo que este se encontra dentro da gama referida pelo fabricante (< 1 μV_{PP}). De notar que a junção da placa de desenvolvimento AFE ADS1299 com a placa do Arduino UNO através da placa PCB desenvolvida no laboratório da UMa não influenciou o nível de ruído no protótipo II, sendo este apropriado para a obtenção de sinais fisiológicos.

7.2 Obtenção de Ondas Geradas Internamente pelo ADS1299 com o Protótipo II

Nesta secção pretendeu-se verificar a capacidade do protótipo II gerar ondas internas através do dispositivo ADS1299. O procedimento realizado para a obtenção dessas ondas foi o mesmo que foi realizado com o protótipo I, seguindo a configuração dos registos

apresentada no Anexo C, dando origem a quatro ondas quadradas diferentes que podem ser visualizadas na Figura 7.2.

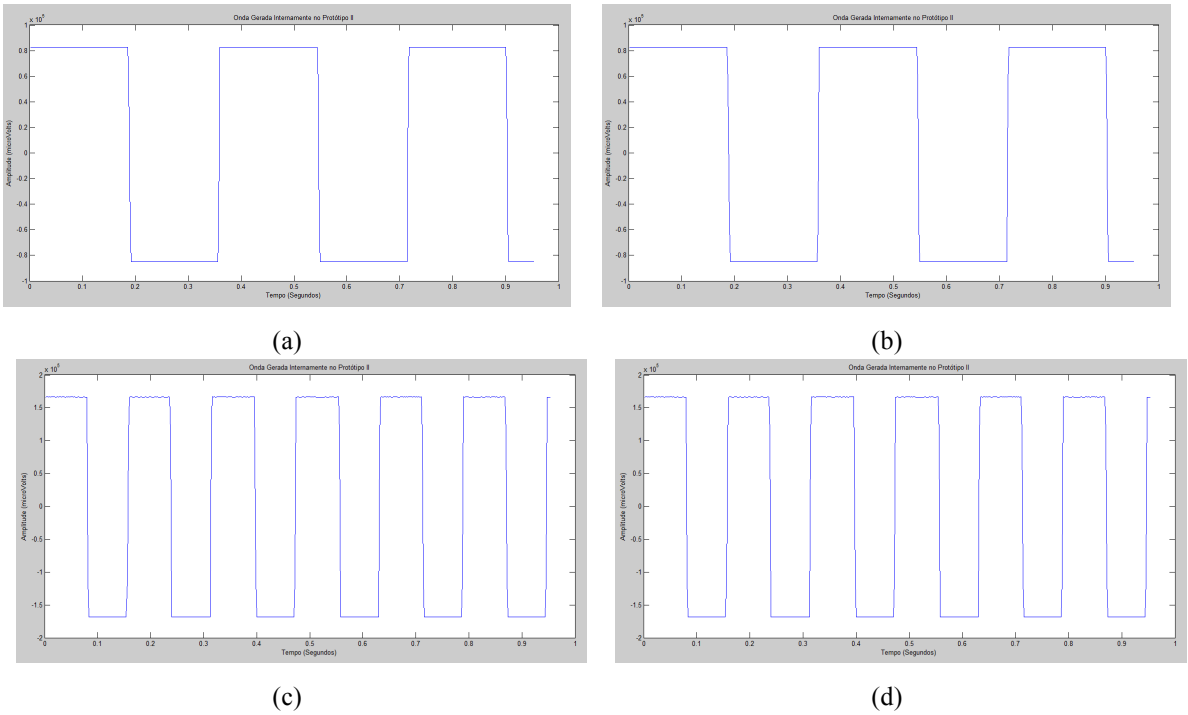


Figura 7.2 – Ondas quadradas geradas internamente no protótipo II pelo dispositivo ADS1299: (a) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (b) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{20}$; (c) Amplitude (VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$; e (d) Amplitude 2x(VREFP-VREFN)/2,4 mV e Frequência $f_{CLK} = 2^{21}$.

Observando a Figura 7.2, pode-se concluir que o protótipo II foi capaz de capturar satisfatoriamente as ondas quadradas geradas internamente pelo dispositivo ADS1299, sendo às mesmas semelhantes as que se obtiveram com o protótipo I.

7.3 Obtenção de Sinais com o Protótipo II utilizando um Gerador de Sinais

De modo a verificar a capacidade de processamento do protótipo II procedeu-se à obtenção de sinais utilizando um gerador de sinais disponível no laboratório. Novamente seguiu-se o procedimento que foi realizado com o protótipo I e a configuração de registos apresentada no Anexo D. Atendendo a istopode-se verificar que o protótipo II conseguiu obter as ondas geradas com recurso a um gerador de sinais, sem qualquer problema, pois o mesmo operava em tempo real, ao contrário do protótipo I.

7.4 Obtenção e Análise de Sinais Fisiológicos com o Protótipo II

Após a verificação do correto funcionamento do protótipo II, o mesmo encontrava-se pronto para a obtenção de sinais fisiológicos e respetiva análise. Com o intuito de avaliar as limitações do protótipo desenvolvido, foram também realizadas aquisições de sinais

fisiológicos com os dispositivos comerciais disponíveis na UMa (apresentados no capítulo III). Como os sinais ECG e EMG são os mais simples de se obterem e manipular, começou-se pela sua aquisição, seguindo-se por último a aquisição dos sinais EEG, por serem mais complexos e ainda por obrigarem à colocação de um capacete.

7.4.1 Resultados e Análise da Aquisição de Sinais de ECG

Para a aquisição ou gravação de sinais ECG, com o protótipo II, teve-se em consideração que os dispositivos comerciais, *PLUX* e *BITalino*, utilizam um elétrodo tripolar (Figura 7.3-a), colocado num autocolante que permite uma melhor aderência à pele (Figura 7.3-b). Este elétrodo tripolar não passa de três elétrodos normais, sendo que dois deles funcionam como elétrodos diferenciais e o terceiro como sinal de referência (elétrodo do meio). Este facto não foi considerado um problema pois o protótipo II permite a obtenção de sinais diferenciais, bastando para isso mudar alguns dos seus registos internos, permitindo uma aquisição de sinais ECG equivalente à dos dispositivos comerciais. Para além das alterações de registos, foi necessário escolher no software *openBCI* a opção de canais diferenciais. Neste caso foram escolhidos o canal 1 e o canal 2, podendo ter sido escolhidos outros canais. As alterações realizadas no protótipo II e a alteração dos registos do ADS1299 são apresentadas no Anexo H.

(a) (b)

Figura 7.3 – Elétrodo ECG Tripolar: (a) Elétrodo dos dispositivos *PLUX* e *BITalino;* e (b) Autocolante que permite uma melhor aderência dos elétrodos à pele.

Tendo sido explicados os procedimentos de obtenção de sinais ECG com o protótipo II, resta dizer que o eletrodo tripolar é colocado na zona ventricular V2 (capítulo 2) do utilizador como se pode observar na Figura 7.4.

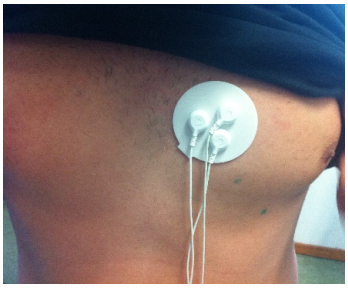


Figura 7.4 – Posicionamento do elétrodo tripolar na zona ventricular V2 do utilizador.

O fabricante do ADS1299 recomenda a utilização de um quarto elétrodo com a função de massa, colocado na perna direita (tornozelo) do utilizador, tal como se pode observar na Figura 7.5, sendo esta uma ligação opcional, que ajuda a reduzir o ruído envolvente no sinal ECG.

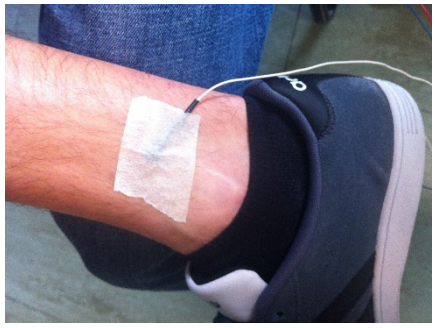
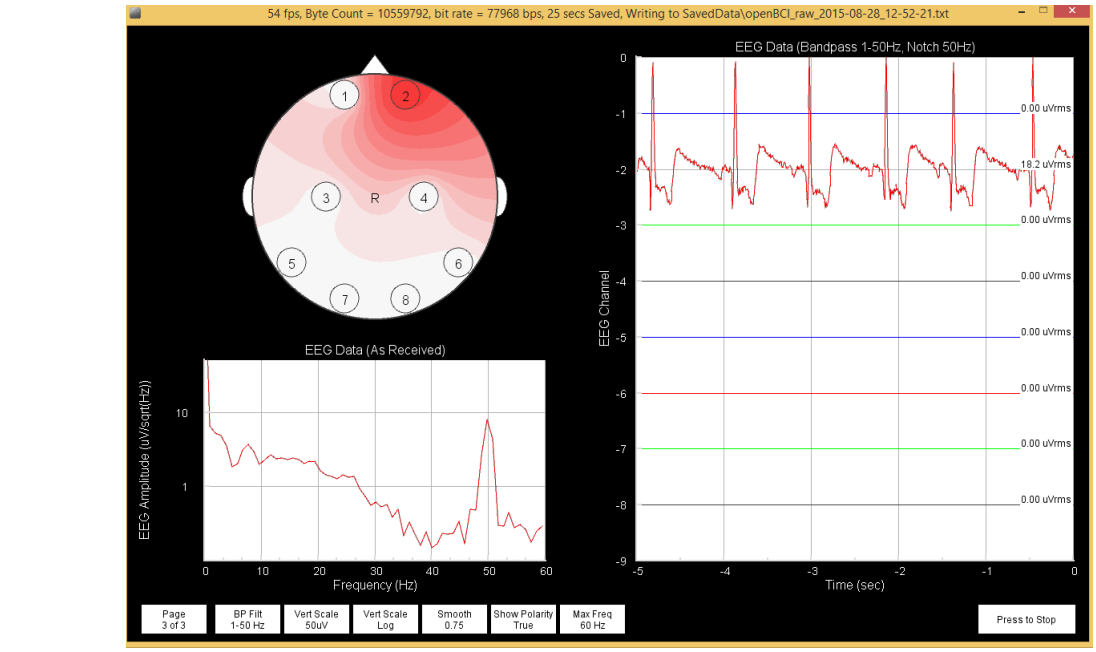


Figura 7.5 – Elétrodo opcional colocado na perna direita (tornozelo) do utilizador, com a função de reduzir o ruído do sinal ECG.

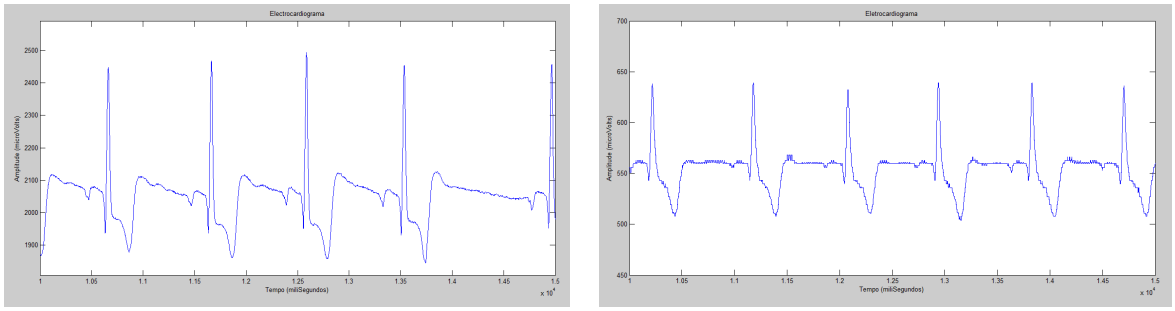
De mencionar que a cada elétrodo foi adicionado um pouco de gel condutivo eletrólito, permitindo uma maior condução entre os mesmos e a pele do utilizador.

Após a correta ligação e posicionamento do elétrodo tripolar e do quarto elétrodo no utilizador, procedeu-se à obtenção dos sinais ECG com o prototipo II e de seguida com os dispositivos, *PLUX* e *BITalino.* Durante a aquisição dos sinais o utilizador encontrava-se em completo repouso, sentado corretamente numa cadeira, durante 2 minutos a olhar para frente. O autocolante dos elétrodos foi mantido fixo no peito do utilizador, sendo apenas alterados os elétrodos correspodentes de cada dispositivo, com o objetivo de se obterem sinais de ECG equivalentes com todos os dispositivos.

Para a visualização dos sinais ECG com o protótipo II, em tempo real, utilizou-se o software *openBCI*, apresentado no capítulo 6. Para os dispositivos *PLUX* e *BITalino*, utilizou-se o software *openSignals*, no entanto este software ainda apresenta uma baixa resolução na apresentação dos dados em tempo real, não sendo apresentada a resolução efetiva destes dispositivos, pelo que os dados obtidos foram manipulados na ferramenta *MATLab* em pós-processamento, com o objetivo de poderem ser comparados com os obtidos com o protótipo II. Na Figura 7.6, podem ser visualizados os sinais de ECG que se obtiveram com o protótipo II e os dois dispositivos comerciais.



(a)



(b) (c)

Figura 7.6 – Sinais de ECG obtidos com: (a) Protótipo II; (b) Dispositivo *PLUX;* e (c) Dispositivo *BITalino.*

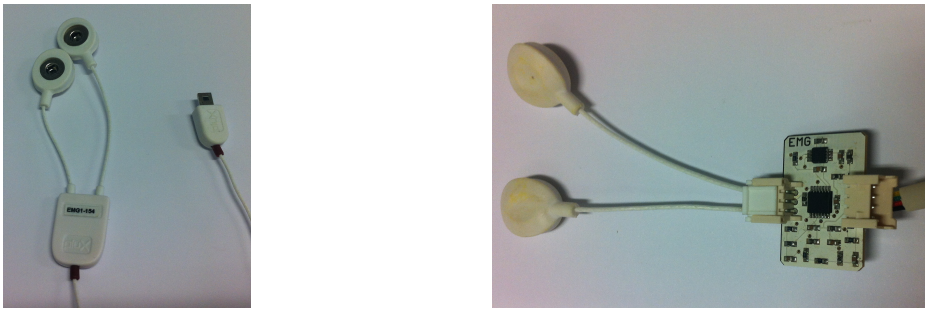
Analisando a Figura 7.6, pode-se concluir que o protótipo II e os dispositivos comerciais conseguem a obtenção de sinais de ECG de forma satisfatória, pois os traçados apresentam o tipo de onda comum de um sinal de ECG completo (

Figura 2.14). Refira-se que num sinal de ECG o complexo QRS é o mais importante, sendo que a onda R permite determinar a frequência cardíaca, e que tanto o protótipo II, como os dispositivos comerciais conseguem obter com exatidão esse complexo. Outro factor de importância é a visualização das ondas P e T, sendo que no protótipo II e no dispositivo *PLUX* essas ondas apresentam grande resolução, pelo que as mesmas se consegue-se distinguir sem qualquer problema. No entanto não se pode dizer o mesmo do sinal que se obteve com o dispositivo *BITalino*, que neste caso apresentava pouco detalhe na visualização dessas ondas o que se deve à baixa resolução do dispositivo (10 bits).

7.4.2 Resultados e Análise da Aquisição de Sinais de EMG

Para a aquisição de sinais de EMG teve-se em conta que os dispositivos comerciais, *PLUX* (Figura 7.7-a) e *BITalino* (Figura 7.7-b), utilizam um par de elétrodos superficiais,

configurados em modo diferencial, de modo a obter diferença de potencial entre dois pontos. Na obtenção de sinais de EMG com o protótipo II, também foram utilizados dois elétrodos configurados em modo diferencial, permitindo uma aquisição equivalente à dos dipositivos comerciais, tal como aconteceu no caso da obtenção dos sinais de ECG. É de salientar que não foi necessária a modificação dos registos internos do protótipo II, pois estes já se encontravam configurados em modo diferencial desde a obtenção dos sinais de ECG.

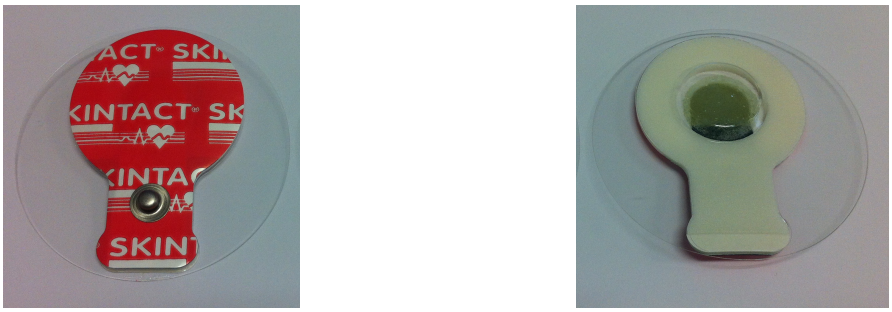


(a) (b)

Figura 7.7 – Elétrodos EMG superficiais: (a) Dispositivo *PLUX* e (b) Dispositivo *BITalino.*

De referir que para a aquisição dos sinais de EMG, com o protótipo II, foram utilizados os canais 1 e 2. Desta vez não será apresentada a ligação física, pois é similar à da secção anterior, à exceção do elétrodo de referência, que não é utilizado neste tipo de aquisição.

Para facilitar a obtenção dos sinais de EMG, utilizaram-se autocolantes especiais que permitem a ligação dos elétrodos numa das faces (Figura 7.8-a) e em que a outra permite a aderência do elétrodo à pele do utilizador (Figura 7.8-b), através de um gel condutivo e eletrólito que melhora a condução do mesmo.



(a) (b)

Figura 7.8 – Autocolantes dos elétrodos de EMG: (a) Face que permite a ligação do elétrodo e (b) Face que permite aderir à pele do utilizador.

Para a obtenção dos sinais de EMG o par de elétrodos foram colocados no músculo *bíceps* do utilizador, como se pode observar na Figura 7.9. Durante a aquisição destes sinais o utilizador encontrava-se a contrair/relaxar o músculo *bíceps* em intervalos de aproximadamente 3 segundos, sentado corretamente numa cadeira, durante 2 minutos a olhar para frente. Tal como na aquisição dos sinais ECG, na obtenção dos sinais de EMG também mantiveram os autocolantes dos elétrodos fixos no músculo *bíceps* do utilizador.

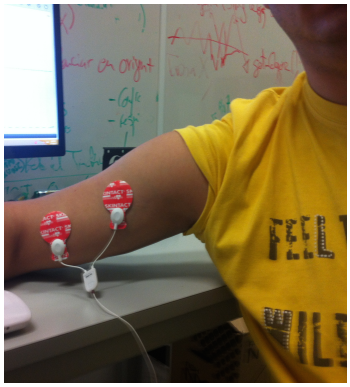
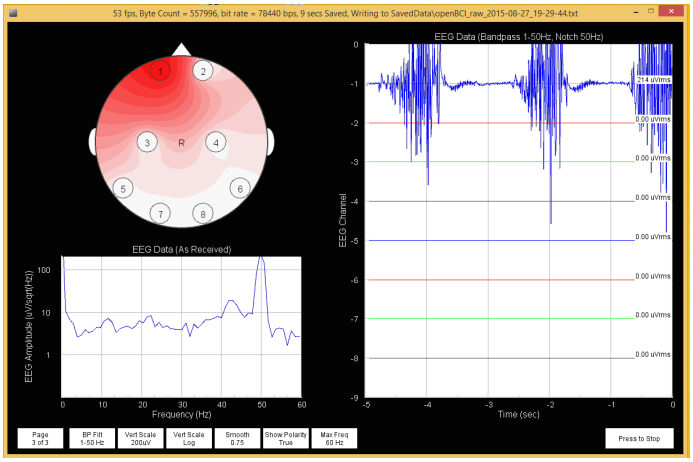
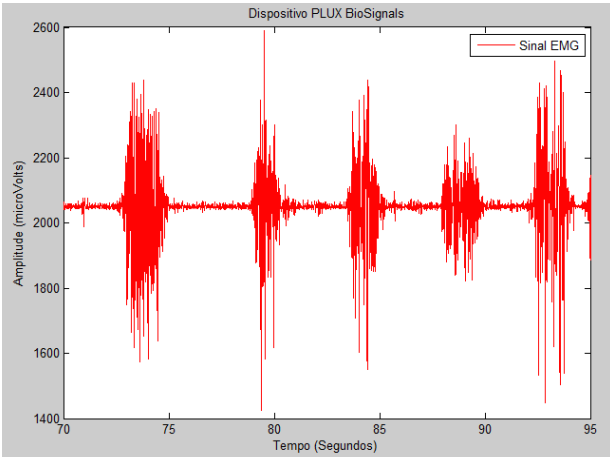


Figura 7.9 – Colocação dos elétrodos de EEG no músculo *bíceps* do utilizador, para a obtenção de sinais de EMG.

Na Figura 7.10-a pode-se visualizar o resultado da obtenção dos sinais de EMG com o protótipo II e na Figura 7.10-b, encontra-se o resultado obtido com o dispositivo *PLUX*. De salientar que não se obtiveram resultados para o dispositivo *BITalino*, pois o módulo de aquisição deste tipo de sinal encontrava-se danificado.



(a)



(b)

Figura 7.10 – Sinais de EMG: (a) Obtido com o Protótipo II através do software *openBCI* e (b) Obtido com o dispositivo *PLUX* e pós-processamento com a ferramenta *MATLab*.

Analisando a Figura 7.10, pode-se concluir que o protótipo II e o dispositivo comercial conseguem a obtenção de sinais de EMG de forma satisfatória, pois os traçados apresentam o tipo de onda comum de um sinal de EMG. No entanto no dispositivo *PLUX*, pode-se verificar que a contração/relaxamento do músculo *bíceps* apresenta uma duração superior ao obtido com o protótipo II. Isto deve-se à menor capacidade de processamento do próprio dispositivo, ou seja, o mesmo tem um tempo de reação mais lento do que o do protótipo II, sendo uma desvantagem deste dispositivo em relação ao desenvolvido neste projeto.

7.4.3 Resultados e Análise da Aquisição de Sinais EEG

A aquisição de sinais de EEG foi dividida em duas partes. A primeira consistiu na visualização de alguns artefactos que afectam os sinais de EEG, permitindo verificar o correto funcionamento do protótipo II, na obtenção desse tipo de sinal. A segunda parte, um pouco mais complexa, consistiu na obtenção de sinais de EEG utilizando o método denominado de *Motor Imagery*, que é atualmente utilizado na área de treino mental e de reabilitação motora quando adicionado ao método *Motor Execution*. De referir que em ambas as partes foi utilizado o protótipo II como dispositivo de obtenção dos sinais de EEG, no entanto para a segunda parte, foram também utilizados outros dois dispositivos comerciais, nomeadamente o *g.tec MOBIlab+* e o *Enobio NE* (apresentados no capítulo 3), com o intuito de avaliar as limitações do protótipo desenvolvido. Os procedimentos de aquisição mencionadas anteriormente serão apresentados de seguida:

Visualização de Artefactos na Obtenção de Sinais de EEG com o Protótipo II: Os artefactos que ocorrem na obtenção de sinais de EEG podem dificultar a sua análise. Entenda-se como artefacto um sinal que não é gerado pela atividade cerebral mas sim por outros responsáveis, tais como a fala, gerando movimentos da musculatura facial, movimentos de diferentes músculos do corpo humano, movimento dos olhos de um lado para outro ou até mesmo o piscar dos olhos. Para reduzir estes artefactos, grande parte dos estudos relacionados com a obtenção de sinais de EEG são realizados evitando estes factores. No entanto neste projeto o facto de existirem alguns desses artefactos não é um problema.

Os artefactos ocorrem em todas as divisões do cérebro humano as quais foram apresentadas no capítulo II. De modo a se poderem visualizar alguns dos artefactos mencionados anteriormente foi decidido distribuir os 8 canais disponíveis no protótipo II em modo unipolar, de forma a abranger todas as divisões do cérebro humano, equitativamente pelos dois hemisférios (esquerdo e direito), seguindo o sistema 10-20.

O posicionamento dos elétrodos foi o seguinte: 2 elétrodos na zona frontal-central (Fc5 e Fc6), 2 elétrodos na zona central-pariental (Cp5 e Cp6), 2 elétrodos na zona temporal (T7 e T8) e finalmente os últimos 2 elétrodos na zona occipital (O1 e O2). Na Figura 7.11, pode-se visualizar o posicionamento escolhido para a colocação dos elétrodos, estando assinalados por cor cada região do cérebro. As alterações realizadas no protótipo II e a alteração dos registos do ADS1299 são apresentadas no Anexo G.

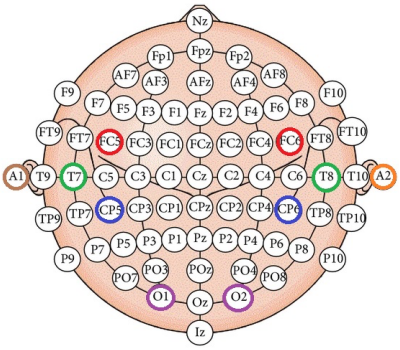
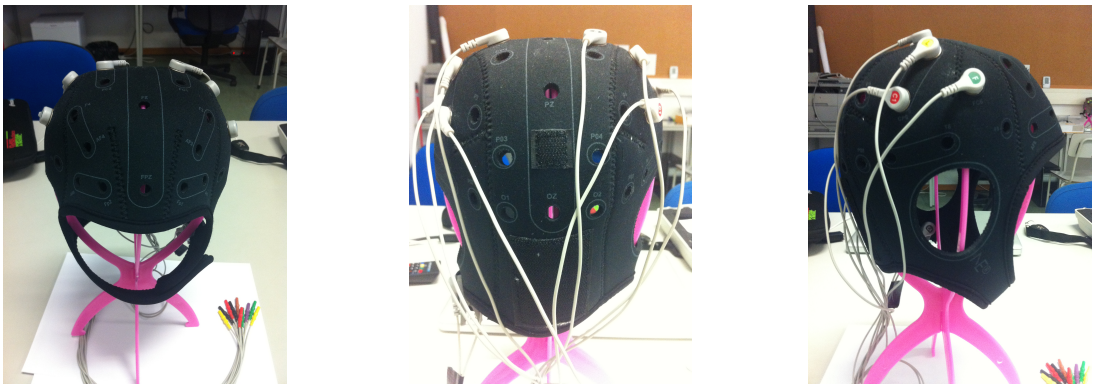


Figura 7.11 – Posicionamento dos elétrodos para a visualização de artefactos dos sinais de EEG.

Foram utilizados ainda outros dois elétrodos, o primeiro com a função de referência comum (cor laranja) e o outro com a função de *ground* (cor castanho). Estes elétrodos devem ser colocados num local onde a atividade fisiológica seja quase inexistente, pelo que foram colocados no lóbulo da orelha esquerda do utilizador. Na Figura 7.12, é possível visualizar o clip que permite a colocação dos elétrodos de referência e de *ground* no lóbulo da orelha do utilizador.

Figura 7.12 – Clip que permite a colocação dos elétrodos de referência e de *ground* no lóbulo da orelha do utilizador.

Tendo sido escolhido o posicionamento dos 8 elétrodos, utilizou-se uma touca mais conhecida por capacete de EEG, a qual facilita a aquisição de sinais (Figura 7.13), mantendo os elétrodos fixos nas respetivas posições.



(a) (b) (c)

Figura 7.13 – Capacete de adquisição de sinais de EEG: (a) Visto da parte de frente; (b) Visto da parte de trás; e (c) Visto de um dos lados.

Em todas as obtenções de sinais fisiológicos, é muito importante o correto posicionamento e respetiva aderência do elétrodo ao utilizador, mas no caso dos sinais de EEG esse factor tem uma importância ainda maior, pois a colocação do capacete na cabeça do utilizador torna o processo mais complexo e demorado, pois um pequeno desvio (< 1 cm) do posicionamento do mesmo, pode implicar estar a abranger uma outra divisão do cérebro, originando sinais não esperados ou até erróneos.

Assim, começou-se pela colocação do capacete na cabeça do utilizador, sendo posteriormente colocados os elétrodos, com um pouco de gel condutivo e eletrólito, ajudando a fixar os mesmos e também para melhorar a condução entre o elétrodo e a zona capilar do utilizador, tendo ainda a função de não permitir a movimentação do elétrodo, encontrando-se o utilizador apto para a obtenção dos sinais de EEG.

De modo a poderem-se visualizar alguns dos artefactos mencionados anteriormente, decidiu-se realizar 4 aquisições de sinais de EEG, com intervalos de 1 minuto de descanso entre elas. Estas aquisições são apresentadas de seguida:

A primeira consistiu na obtenção de sinais de EEG, sendo que o utilizador encontrava-se em estado de repouso, sentado corretamente numa cadeira a olhar para frente. Este tipo de aquisição também é conhecida pelo nome de *baseline*, na área da fisiologia. Na Figura 7.14, pode-se observar o resultado que se obteve com o protótipo II, após o utilizador se encontrar no referido estado durante 5 minutos.

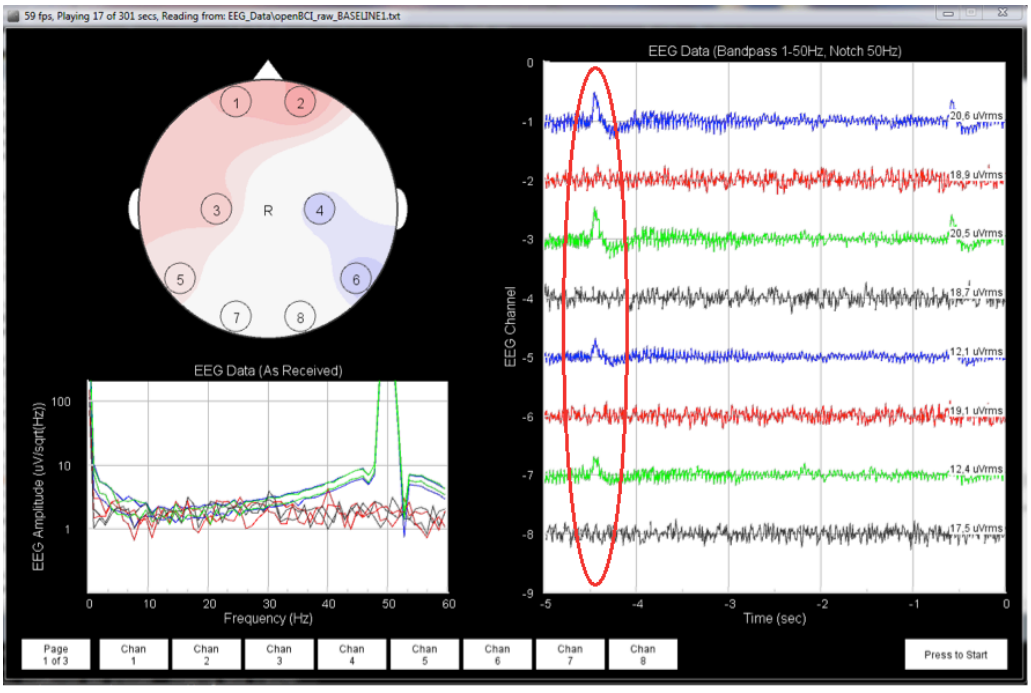


Figura 7.14 – Sinal EEG obtido com o protótipo II, no estado de *baseline*.

Analisando o resultado que se obteve na Figura 7.14, pode-se dizer que o nível de sinal se encontrava adequado, apresentado o tipo de ondas esperadas para este tipo de estado. É Ainda de salientar que apesar do utilizador se encontrar em completo repouso, o mesmo não consegue evitar o piscar dos olhos durante o decorrer da aquisição, sendo considerado isto um

artefacto no sinal de EEG, denominado de *blink*, o qual estará presente em todas as obtenções de sinais de EEG. Este artefacto pode ser visualizado na Figura 7.14, realçado através de um circulo em cor vermelha.

A segunda aquisição compreendeu a obtenção do sinal de EEG de modo a se poder visualizar o estado de *blink,* apesar deste ser visualizado em todas as obtenções, o objetivo desta vez era verificar a performance do protótipo II na sua detecção. Durante um 1 minuto o utilizador piscava os olhos de 5 em 5 segundos, podendo ser visualizado o resultado obtido na Figura 7.15, sendo apresentado num circulo de cor vermelha o estado de *blink*.

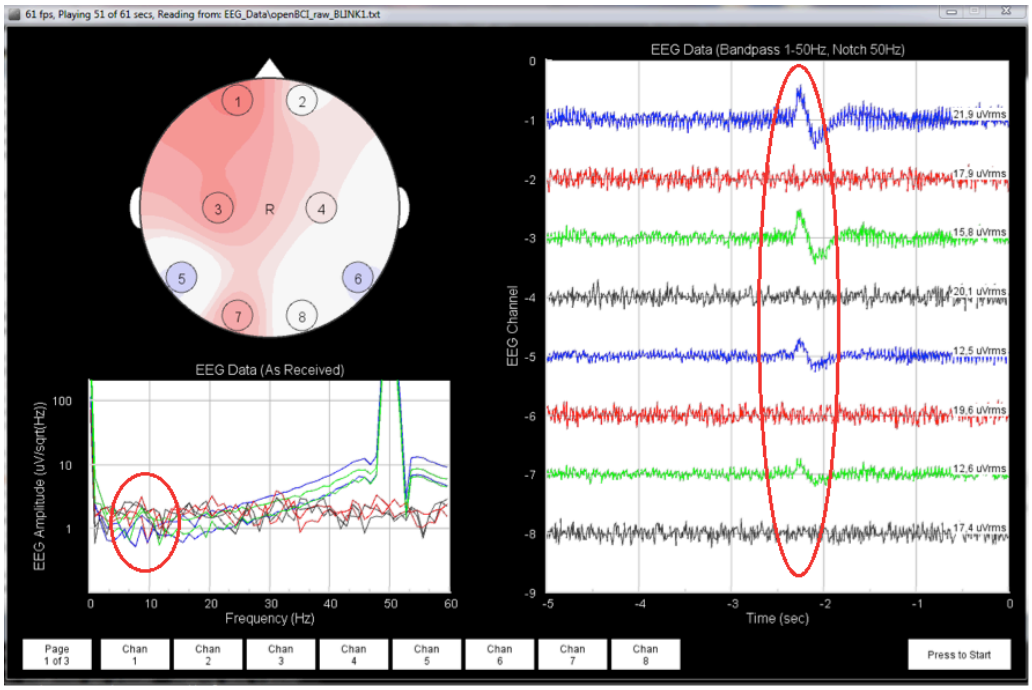


Figura 7.15 – Sinal EEG obtido com o protótipo II, no estado de *blink.*

A terceira aquisição consistiu em verificar o artefacto que é originado no sinal de EEG quando o utilizador decide movimentar os olhos, nomeadamente de cima para baixo, em intervalos de 5 em 5 segundos, durante um minuto. Este artefacto é mais conhecido como *roll eyes*, e pode ser visualizado na Figura 7.16, indicando-se o estado do posicionamento dos olhos do utilizador, a cor vermelho indicando o artefacto de *blink*, a verde indicando de que o utilizador se encontrava a olhar para cima e a laranja indicando que o mesmo se encontrava a olhar para baixo.

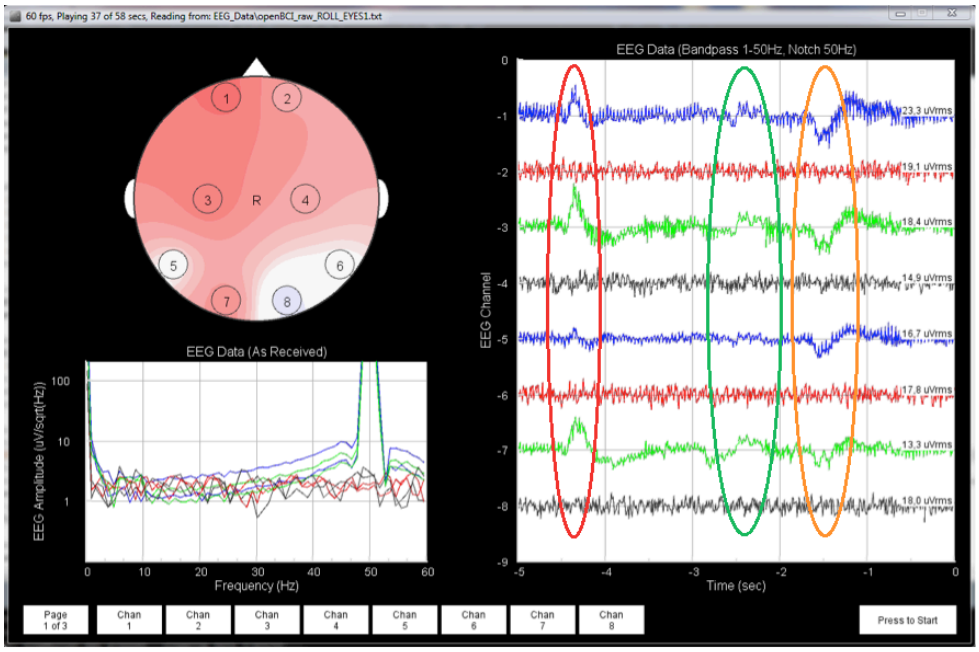


Figura 7.16 – Sinal EEG obtido com o protótipo II, no estado de *roll eyes.*

Observando a Figura 7.16 pode-se concluir que quer movimentos voluntários quer não voluntários dos olhos afectam o sinal de EEG, criando os chamados artefactos nos EEG.

Por outro lado, a musculatura facial é movimentada em diversos casos, por exemplo, quando se movimenta a língua, abre ou fecha a boca e também ao falar, originando um outro tipo de artefacto, denominado de *open/close mouth*. Com o intuito de se visualizar este artefacto o utilizador abria/fechava a boca, em intervalos de 5 em 5 segundos, durante 1 minuto. O resultado obtido com o protótipo II encontra-se na Figura 7.17.

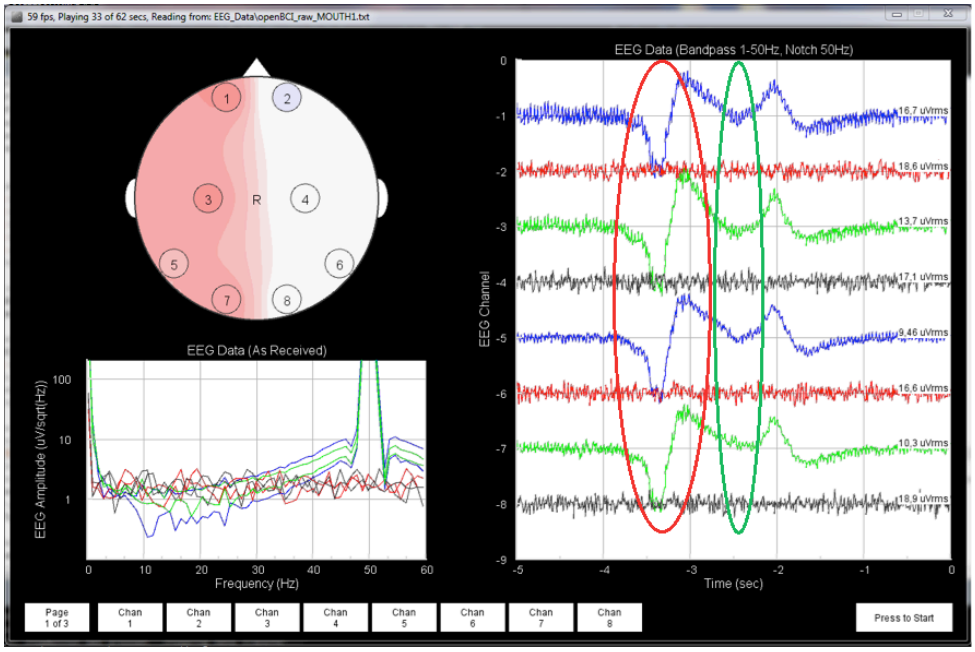
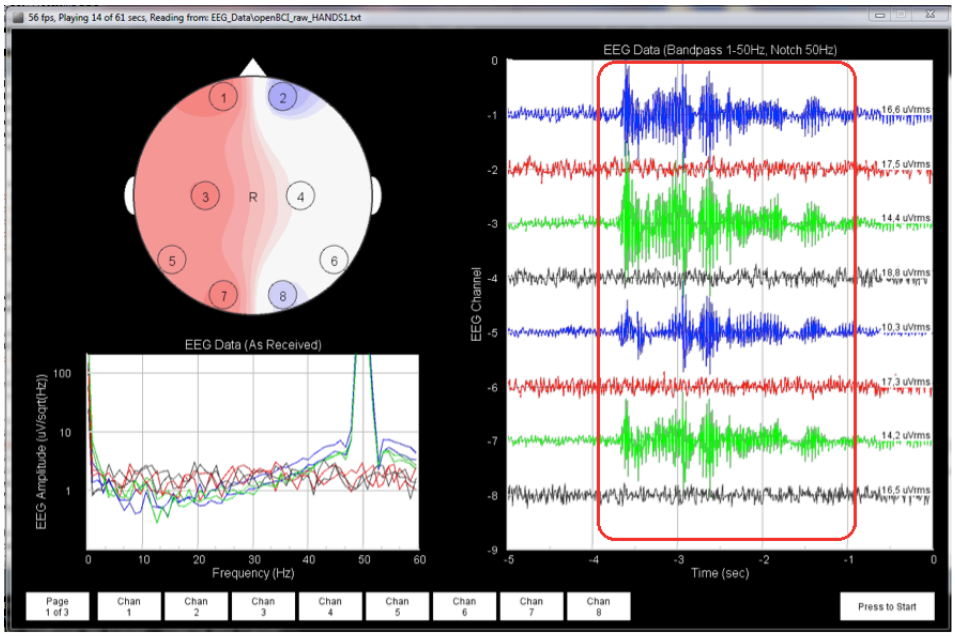


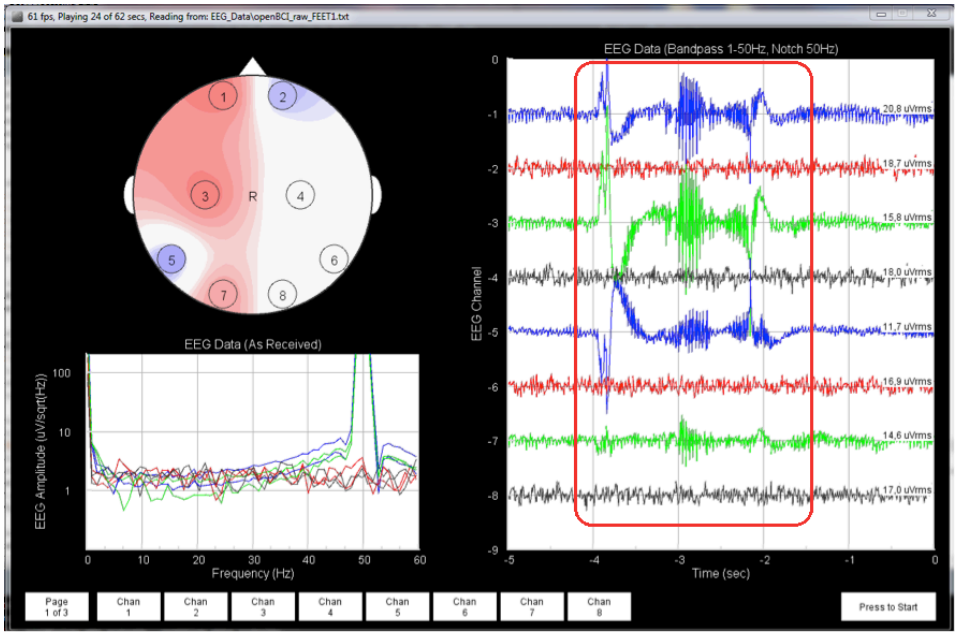
Figura 7.17 – Sinal EEG obtido com o protótipo II, no estado de *open/close mouth.*

Viu-se anteriormente que o sinal de EEG é afectado pelo movimento da musculatura facial. Então torna-se óbvio que qualquer movimento muscular por parte do utilizador durante

o período de obtenção do sinal de EEG terá o mesmo efeito, originando diferentes artefactos. Atendendo a isto as duas últimas obtenções de sinais de EEG com o protótipo II desta primeira parte, consistiram em gerar movimentos musculares com as mãos e com os pés, respetivamente, em intervalos novamente de 5 em 5 segundos, durante 1 minuto. Os resultados obtidos em ambos testes, podem ser visualizados na Figura 7.18.



(a)



(b)

Figura 7.18 – Sinal EEG que obteve-se com o protótipo II: (a) No estado de *move hands;* e (b) No estado de *move feet*.

Analisando a Figura 7.18, pode-se verificar que estes artefactos são semelhantes aos sinais obtidos no estudo do EMG, no entanto apresentando menor amplitude.

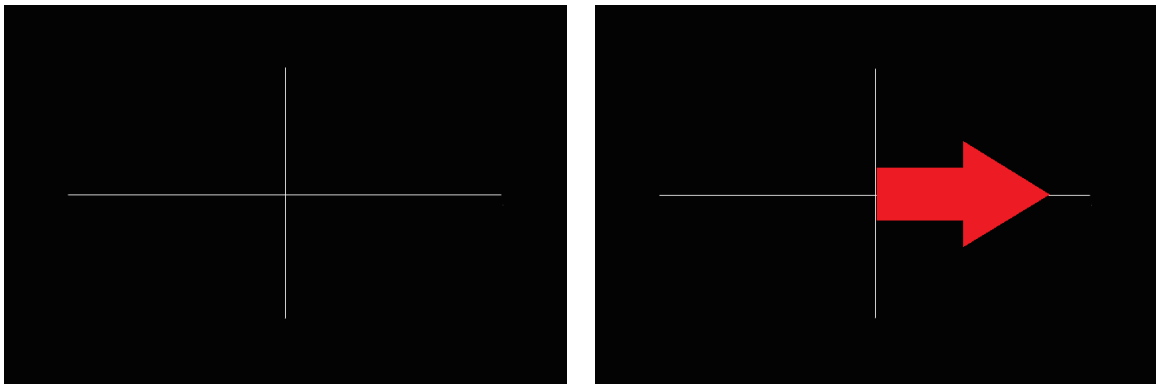
Após terem sidos apresentados alguns sinais típicos de EEG, estudando os respetivos artefactos, verificou-se o correto funcionamento do protótipo II na obtenção deste tipo de medidas fisiológicas.

Obtenção de sinais de EEG utilizando o Método *Motor Imagery*: Hoje em dia os sinais de EEG não só permitem determinar problemas neurológicos, como também ajudam na área da reabilitação em pessoas, como por exemplo nos casos das sequelas motoras que ficam em determinadas pessoas após sofrerem um AVC. No entanto na maioria desses casos o problema não é muscular, mas sim devido a uma parte do cérebro, que ficou danificada. Atualmente são utilizados diversos métodos que ajudam a contornar essas sequelas. O *Motor Imagery* é um dos métodos que pretende, através de pequenos exercícios mentais, estimular os sectores danificados do cérebro, tentando que o mesmo recupere o movimento muscular.

O método de *Motor Imagery* consiste basicamente em colocar o utilizador a visualizar diferentes padrões de imagens e/ou vídeos num ecrã, para este realizar uma determinada ação. Atendendo a isto, utilizou-se o software *openVibe*, o qual permite o desenvolvimento e utilização deste tipo de padrões, através de sistemas de blocos criados pelas comunidades *OpenSource*, de modo a se poder avaliar o desempenho do protótipo II com a utilização deste tipo de métodos, sendo estes posteriormente comparados com os resultados obtidos com os dispositivos comerciais, *g.tec MOBIlab+* e o *Enobio NE*, os quais conseguem ótimos rendimentos. De salientar que a ligação do protótipo II ao software *openVibe* foi realizada em conjunto com o aluno Athanasios Vourvopoulos, o qual se encontrava a trabalhar com este software na sua tese de Doutoramento.

A fase de avaliação do protótipo II consistiu em duas etapas que serão apresentadas abaixo, sendo realizados testes em 8 utilizadores diferentes, aos quais foram solicitados alguns dados demográficos (Anexo J), com o intuito de serem introduzidos no software *openVibe*, o que influenciava o resultado obtido. De referir que estes 8 utilizadores também foram utilizados para a obtenção de resultados com os dispositivos comerciais. De modo a não influenciar os resultados, realizaram-se as medições de forma aleatória em cada utilizador, no que diz respeito à ordem dos dispositivos de medição.

Na primeira etapa foi utilizado um dos padrões mais básicos, que consiste em mostrar ao utilizador setas, que surgem aleatoriamente para esquerda ou para a direita num ecrã (Figura 7.19), sendo que o utilizador deverá através do seu pensamento tentar movimentar o braço esquerdo ou o direito (sem realmente o movimentar), consoante o sentido da seta. De referir que se escolheram os braços, podendo ser outros membros do corpo humano, como os pés, mãos, dedos, etc., dependendo isto de cada caso clínico. Este tipo de padrão é conhecido por etapa de treino do utilizador, sendo que o resultado obtido é expresso em percentagem (0% - 100%), baseado no número de vezes em que o pensamento conseguiu acompanhar o sentido das setas.



(a) (b)

Figura 7.19 – Interface mostrada ao utilizador na etapa de treino: (a) Ecrã inicial; e (b) Ecrã mostrando uma seta para a direita.

De referir que a utilização deste método envolve um posicionamento especifico dos elétrodos, de modo a abranger determinadas zonas do cérebro que permitem a leitura da atividade do pensamento do utilizador. O posicionamento dos elétrodos foi o seguinte (Figura 7.20): 4 elétrodos no hemisfério esquerdo (FC5, CP5, C3 e C1) e 4 elétrodos no hemisfério direito (FC6, CP6, C4 e C2). Em relação aos elétrodos de referência e de massa, estes foram colocados no lóbulo da orelha esquerda e direita respetivamente, da mesma forma que na visualização de artefactos na obtenção de sinais de EEG.

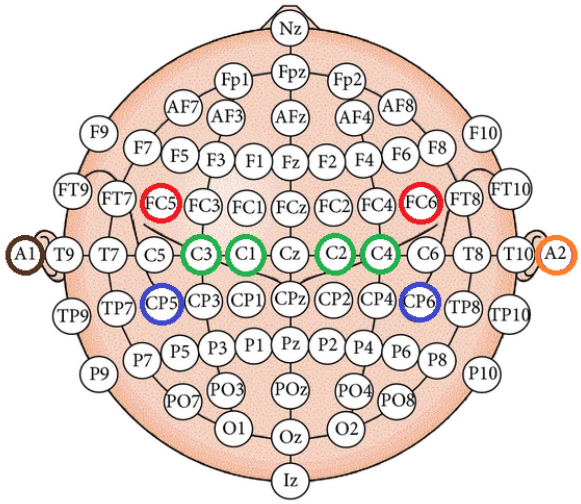
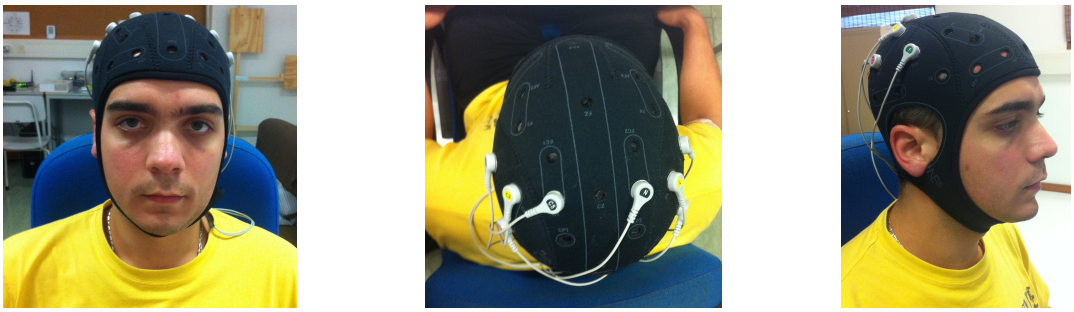


Figura 7.20 – Posicionamento dos elétrodos para a utilização do método *Motor Imagery*.

Posteriormente procedeu-se à colocação do capacete em cada um dos utilizadores, atendendo ao posicionamento dos elétrodos apresentado anteriormente, o qual se pode observar na Figura 7.21.



(a) (b) (c)

Figura 7.21 – Capacete de EEG colocado na cabeça do utilizador: (a) Visto de frente; (b) Visto de cima; e (c) Visto do lado direito.

Após a colocação do capacete no utilizador, decidiu-se verificar o nível de ruído do sinal, sendo este adequado. Cada treino teve uma duração de aproximadamente 7 minutos, em que os primeiros 30 segundos serviam para a concentração do utilizador e após esse período começavam a surgir as setas aleatórias no ecrã. Cada utilizador dispôs de aproximadamente 15 minutos do seu tempo para cada dispositivo, tendo em consideração a montagem/desmontagem do capacete de EEG. Os resultados que se obtiveram após as 3 sessões de treino de cada utilizador, podem ser visualizados na Tabela 7.1. No Anexo K podem-se visualizar alguns dos utilizadores com o capacete de EEG dos dispositivos fisiológicos em causa.

Tabela 7.1 – Resultados obtidos na etapa de Treino com o Protótipo II.

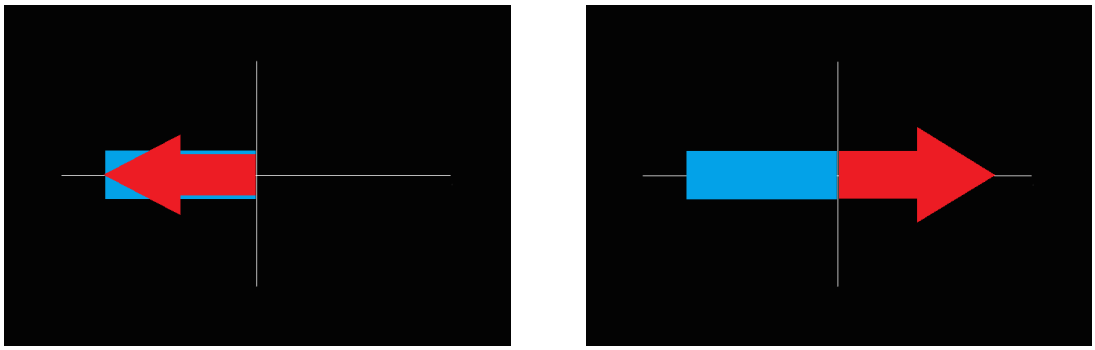
Utilizador	Protótipo II (%)	*Enobio NE* (%)	*g.tec MOBILab+* (%)
1	51,02	67,6	63,1
2	55,96	67,34	66,12
3	57,5	67,19	61,37
4	58,36	73,16	67,39
5	56,12	68,06	65,21
6	57,19	66,93	57,34
7	58,67	71,68	70,66
8	54,84	60,76	73,52
Média	56,21	67,84	65,59

Analisando os resultados da Tabela 7.1, pode-se verificar que com o protótipo II conseguiu-se um valor médio de 56,20 %, ficando aproximadamente 10 % abaixo face à média que se obteve com os dispositivos comerciais. No entanto apesar de esta diferença já ser considerável quando se avalia a performance de dispositivos electrónicos, pode-se concluir que o protótipo II, é um dispositivo valido para a obtenção de medidas fisiológicas, atendendo à relação preço/performance, pois os dispositivos comerciais tem um preço muito superior.

Anteriormente foi referido que os dispositivos comerciais tinham uma elevada performance na realização deste tipo de método, nomeadamente na ordem dos 90 %, o que

não foi aqui verificado. Isto deve-se ao facto desses resultados serem baseados em pessoas treinadas diariamente durante um intervalo considerável, normalmente superior a um mês. Com base no que foi referido anteriormente, pode-se dizer que os resultados que se obtiveram são meramente ilustrativos, demostrando que o protótipo II consegue lidar com este tipo de método sendo provável que a sua performance atinja valores rondando os 80 %.

A segunda parte da avaliação é denominada etapa online e consistiu em visualizar o resultado do treino da primeira parte, mas com a introdução de barras, de cor azul, que indicam se o utilizador fez a escolha certa através do seu pensamento, caso as barras coincidam com o sentido da seta (Figura 7.22). No entanto o utilizador deverá realizar o mesmo procedimento da primeira parte, pois o resultado que se obtém após concluída a etapa online, é dado pela média dos resultados que se obtiveram em ambas as partes, sendo este igualmente expresso em percentagem.



(a) (b)

Figura 7.22 – Interface mostrada ao utilizador na etapa online: (a) Seta e barra azul coincidindo, indicando o acerto do utilizador; e (b) Seta e barra azul com sentidos opostos, indicando o erro do utilizador.

Os resultados que se obtiveram após as 3 sessões online de cada utilizador, podem ser visualizados na Tabela 7.2.

Tabela 7.2 – Resultados obtidos na etapa Online, com o Protótipo II.

Utilizador	Protótipo II (%)	*Enobio NE* (%)	*g.tec MOBILab+* (%)
1	51,8	49,69	50,1
2	50,95	47,93	43,02
3	49,97	45,51	49,9
4	59,37	59,3	51,94
5	50,26	41,91	54,74
6	45,08	56,56	57,59
7	50,57	38,41	51,53
8	46,37	52,76	56,33
Média	50,55	49,01	51,89

Atendendo aos resultados que apresentados na Tabela 7.2, pode-se dizer que as médias do protótipo II e dos dispositivos comerciais foram semelhantes. Novamente os valores ficaram abaixo do esperado, devendo-se isto ao reduzido treino dos utilizadores já referido

anteriormente. Os resultados do teste online são influenciados por parte do utilizador, quando o mesmo se encontra a visualizar o resultado das opções tomadas no teste de treino e ao mesmo tempo tenta não se enganar no teste online. Por esta razão os resultados da média do protótipo II foi equivalente aos dos dispositivos comerciais.

7.5 Conclusões do Capítulo

Neste capítulo foram apresentados os resultados obtidos após se realizarem alguns testes de modo a avaliar o desempenho do protótipo II desenvolvido e ainda para o comparar com os dispositivos comerciais apresentados no final do Capítulo 3.

O primeiro teste realizado com o protótipo II consistiu na medição do ruído, factor de elevada importância quando se trabalha na área dos dados fisiológicos, obtendo-se um valor de ruído que encontrava-se dentro da gama referida pelo fabricante.Também se verificou com o protótipo II a capacidade do ADS1299 gerar ondas internas, sendo estas obtidas satisfatoriamente, através da manipulação de dois parâmetros, nomeadamente da amplitude e da frequência. Foi também utilizado um gerador de sinais de laboratório com o intuito de se obterem sinais com o modo de aquisição *Continuous*, sendo que se verificou que o protótipo II conseguia a aquisição de sinais em tempo real corretamente, tendo sido ultrapassado o problema que se verifico com o protótipo I.

O teste de obtenção de sinais fisiológicos foi dividido em 3 fases, a primeira consistiu na obtenção de sinais de ECG, a segunda nos sinais de EMG e a última compreendeu a obtenção de sinais de EEG. Na obtenção de sinais de ECG pôde-se concluir que o protótipo II conseguia a sua aquisição de modo satisfatório, sendo equivalente aos sinais obtidos com dispositivo comercial *BioSignals PLUX* e superior aos do dispositivo *BITalino*, pois viu-se que este apresentava um sinal de ECG com pouca resolução. Em relação aos sinais de EMG é de mencionar que não se conseguiu a obtenção de sinais com o dispositivo *BITalino* pois o módulo de obtenção desse tipo de sinal encontrava-se danificado, sendo que para o protótipo II e para o dispositivo *PLUX,* foram obtidos com sucesso, apresentando uma elevada resolução. Viu-se que a aquisição de sinais de EEG era afetada pela ocorrência de alguns artefactos, os quais surgem em todas as regiões do cérebro, sendo que o protótipo II consegue a sua detecção satisfatoriamente. Os mesmos não foram considerados um problema já que são utilizados no estudo deste tipo de medição fisiológica. Ainda foi avaliado o protótipo II utilizando o método *Motor Imagery*, sendo este dividido em duas etapas, uma de treino e a outra online. O resultado obtido na primeira etapa com o protótipo II foi de menos 10 % face à média que se obteve com os dispositivos comerciais. Na segunda etapa foi obtido um valor semelhante em todos os dispositivos.

Verificou-se que protótipo II é um dispositivo válido para a obtenção de medidas fisiológicas, atendendo à relação preço/performance, pois os dispositivos comerciais tem um preço muito superior.

8 Conclusões e Trabalhos Futuros

Neste capítulo são abordadas as principais conclusões acerca do desenvolvimento e avaliação de um sistema de medição fisiológica de baixo custo. Posteriormente é realizada uma pequena exposição relativamente aos aspetos que podem ser melhorados e desenvolvidos.

8.1 Conclusões

Com a realização deste trabalho verificou-se que apesar de existir uma vasta variedade de dispositivos comerciais que permitem a aquisição das principais medidas fisiológicas, nenhum desses dispositivos possui as características desejadas para o mesmo poder ser considerado um dispositivo de baixo custo, com grande resolução e que permita a medição de diversas medidas fisiológicas, tais como EEG, ECG e EMG, num único dispositivo.

Estudou-se então o AFE ADS 1299, tendo-se concluído que este dispositivo, devido às suas caraterísticas, poderia ser a base que permitiria a elaboração de um protótipo de um dispositivo de medição de grandezas fisiológicas, com um baixo custo e com uma elevada resolução (24 bits/canal).

O protótipo I, que consistiu no estudo do *kit* de desenvolvimento da TI (AFE ADS1299 + DSP MMB0 + software em *LabVIEW*), ficou um pouco além das expectativas, pois após realizados alguns testes foi verificado que apesar do software ser muito intuitivo, fornecer diversas opções de visualização (domínio dos tempos, domínio das frequências e histogramas) e de permitir guardar os dados em diversos formatos, não permitia a aquisição dos dados em tempo real, devido a uma limitação de hardware da DSP MMB0. Concluiu-se assim que o protótipo I, não satisfazia os requisitos iniciais deste trabalho, sendo necessário desenvolver uma outra solução. De mencionar ainda que para chegar à conclusão anterior demorou-se algum tempo, pois a TI afirmava que os dados se conseguiam obter em tempo real através da sua DSP, modificando apenas alguns sistemas de blocos e funções do seu software. Tal facto não foi conseguido com sucesso, sendo que a própria TI atualizou o seu *datasheet* indicando a limitação da DSP.

Posteriormente ao estudo realizado ao protótipo I, verificou-se que era fundamental a presença de quatro módulos para o desenvolvimento de um novo protótipo, o qual foi denominado de protótipo II sendo este o protótipo final deste trabalho. Esses módulos são os seguintes: o módulo de aquisição fisiológica, o módulo do sistema de alimentação, o módulo de comunicação e por último o módulo do sistema de visualização dos dados em tempo real. Outro ponto essencial consistiu na escolha mais adequada dos componentes que constituem as unidades referidas, visto que é um requisito fundamental na construção de um protótipo de medidas fisiológicas de baixo custo.

Atendendo ao anterior a construção/desenvolvimento do protótipo II foi baseada no projeto *open source* desenvolvido pelo grupo *openBCI*, uma vez que esse grupo também utilizou a placa de desenvolvimento AFE ADS1299 da TI. No que concerne à construção do protótipo II refere-se que este foi um processo demorado pois envolveu diversas etapas com o

intuito de desenvolver o mesmo com sucesso. A ligação do Arduino UNO a placa de desenvolvimento AFE ADS1299 via SPI permitiu que os dados de saída do ADS1299 fossem obtidos em tempo real superando a limitação que existia com o protótipo I. O hardware que compõem o protótipo II possibilita a integração de periféricos ao contrario do que acontecia com o protótipo I , sendo que foi adicionado um módulo de comunicação sem fios (*Bluetooth*), para permitir a mobilidade do protótipo II. O sistema de alimentação do protótipo II garante alguma segurança ao utilizador pois é alimentado através de uma *pack* de 8 baterias recarregáveis de 1,2 V, sendo esta uma vantagem face a alimentação do protótipo I, que era através de um transformador AC/DC.

Após a conclusão do protótipo II foram realizados alguns testes para verificar o correto funcionamento do mesmo. O primeiro consistiu na verificação do nível de ruído, sendo confirmando que este se encontrava dentro da gama referida pelo fabricante ($< 1\ V_{PP}$). Também foi testada a capacidade do protótipo II gerar ondas internas através do dispositivo ADS1299, verificando-se que o mesmo as conseguiu obter com elevada resolução e precisão. Para avaliar a capacidade de processamento do protótipo II procedeu-se à obtenção de sinais utilizando um gerador de sinais disponível no laboratório, sendo os mesmos obtidos satisfatoriamente sem qualquer problema, pois o sistema operava em tempo real, ao contrário do protótipo I.

Verificado o correto funcionamento do protótipo II desenvolvido, foram obtidos os sinais fisiológicos, começando-se pelas aquisições de sinais de ECG e de EMG, por serem mais simples de se obter e por último os sinais de EEG, por serem mais complexos e ainda por envolverem a utilização de uma touca de EEG.

Em relação aos sinais de ECG pôde-se concluir que o protótipo II conseguia a sua aquisição de modo satisfatório, sendo equivalente aos sinais obtidos com dispositivo comercial *BioSignals PLUX* e superior aos do dispositivo *BITalino*, pois viu-se que este apresentava um sinal de ECG com pouca resolução. Em relação aos sinais de EMG é de mencionar que não se conseguiu a obtenção de sinais com o dispositivo *BITalino* pois o módulo de obtenção desse tipo de sinal encontrava-se danificado, sendo que para o protótipo II e para o dispositivo *PLUX,* foram obtidos com sucesso, apresentando uma elevada resolução.

No que concerne aos resultados de aquisição de sinais de EEG viu-se que os mesmos eram afetados pela ocorrência de alguns artefactos, os quais surgem em todas as regiões do cérebro, sendo que o protótipo II consegue a sua detecção satisfatoriamente. Os mesmos não foram considerados um problema já que são utilizados no estudo deste tipo de aquisição fisiológica. Ainda foi avaliado o protótipo II utilizando o método *Motor Imagery*, sendo este dividido em duas etapas, uma de treino e a outra online. O resultado obtido na primeira etapa com o protótipo II foi de menos 10 % face a média que se obteve com os dispositivos comerciais, na segunda etapa foi obtido um valor semelhante em todos os dispositivos.

Como conclusão final deste trabalho pode-se afirmar com os resultados que se obtiveram que o protótipo II pode servir como uma ferramenta de ensino e de pesquisa, de baixo custo atendendo à relação preço/performance quando comparados com os dispositivos

comerciais aqui apresentados, facilitando a interação cérebro-computador e permitindo criar paradigmas de treino mental do tipo *neuro-feedback* para pessoas com danos cerebrais.

8.2 Trabalhos Futuros

Relativamente aos objetivos iniciais, o trabalho desenvolvido correspondeu as expectativas. O protótipo II para a aquisições de sinais fisiológicos funciona como esperado, embora exista a necessidade de otimizar o sistema no que diz respeito as seguintes componentes:

Melhoria do hardware a nível de processamento, ou seja, a utilização de um microcontrolador que consiga aproveitar a capacidade máxima de processamento do ADS1299 (16000 SPS) a qual se encontra limitada a um máximo de 250 SPS devido a capacidade de processamento do Arduino UNO.

Melhorar o sistema de alimentação do protótipo II, utilizando baterias de lítio que possuem uma maior autonomia que as pilhas recargáveis convencionais.

Melhorar a interface do software de visualização em tempo real dos dados fisiológicos, permitindo a alteração dos registos do protótipo II em tempo real e ainda permitir realizar zoom nos gráficos dos domínio dos tempos e das frequências.

De modo a visualizar os dados corretamente no pós-processamento, desenvolvimento de algoritmos que permitam a correta filtragem dos dados no pós-processamento.

Referências Bibliográficas

[1] TERESA PAIVA, "Características e registos de algumas variáveis fisiológicas," FML, Dezembro 2006.

[2] JAAKKO MALMIVUO e ROBERT PLONSEY, "Bioelectromagnetism - Principles and Applications of Bioelectric and Biomagnetic Fields," *Oxford University* Press,1995.

[3] R. M. BERNE, M. N. LEVY e B. M. KOEPPEN, Fisiologia, Tradução Brasileira da 5ª edição americana, Elsevier Editora, 2004.

[4] U. WOLF, M. J. RAPOPORT e T. A. SCHWEIZER, "Evaluating the affective component of the cerebellar cognitive affective syndrome," *The Journal of Neuropsychiatry and Clinical Neurosciences,* vol. 21 No. 3, pp. 245-253, 2009.

[5] E. R. KANDEL, J. H. SCHWARTZ e T. M. JESSELL, "Principles of neural science," 2000.

[6] F. H. NETTER, J. A. CRAIG e J. PERKINS, Atlas of neuroanatomy and neurophysiology,, Icon Costum Communications, 2002.

[7] KO KEUN KIM YONG KYU LIM e KWANG SUK PARK,"ECG Recording on a Bed During Sleep Without Direct Skin-Contact," *in IEEE Transactions on Biomedical Engineering*, Abril 2007, vol. 54, no. 4, pp. 718-725.

[8] [8] ARMANDO MORENO, "Anatomofisiologia - Tomo III", Faculdade de Motricidade Humana - Universidade Técnica de Lisboa , Outubro de 1991.

[9] CHR. ZYWIETZ, "A Brief History of Electrocardiography and Electroencephalography - Progress through Technology," *in Biosignal Institute for Biosignal Processing and Systems Research, Hannover*.

[10] E. NIEDERMAYER e F. D. SILVA, Electroencephalography: basic principles, clinical applications and related fields,, 1st Edition: Lippincott, 2005.

[11] J. A. BARROS, C. G. CARVALHAES e J. P. R. F. MENDONÇA, "Recogniton of words from the EEG Laplacian," *Brazilian Journal of Biomedical Engineering,* vol. 21, pp. 143-150, December 2005.

[12] B. HJORTH, "An on-line transformation of EEG scalp potentials into orthogonal source derivations," *Electroencephalography and Clinical Neurophysiology,* vol. 39 No. 5, pp. 526-530, 1975.

[13] N. S. DIAS, J. P. CARMO e P. M. MENDES, "Wireless instrumentation system based on dry electrodes for acquiring EEG Signals," *Medical Engineering & Physics: Elsevier Science,* pp. 1-10, 2012.

[14] N. S. DIAS, Interface cerebro-máquina baseada em biotelemetria e eléctrodos secos,, Universidade do Minho: Tese de Doutoramento em Engenharia Eletrónica Industrial,

2009.

[15] "*Open Signals*". Disponível em: http://www.opensignals.net/

Consultado em 10 de Janeiro de 2015.

[16] S. J. SCHIFF, D. COLELLA e G. M. JACYNA, "Brain chirps: spectrographic signatures of epileptic seizures," *Clinical neurophysiology: Official Journal of the International Federation of Clinical neurophysiology,* vol. 111 No. 6, pp. 953-958, 2000.

[17] R. A. RHOADES e R. G. PFLANZER, Human Physiology, 4th Edition: Brooks Cole, 2002.

[18] J. C. RODRIGUES, Electrocardiografia clinica: principios fundamentais, Lidel, 2008.

[19] D. L. LONGO, A. S. FAUCI e D. L. KASPER, Harrison's principles of internal medicine,, 18th Edition: Mc-Graw-Hill Medical, 2011.

[20] A. DESPOPULOS e S. SILBERNAGL, Color atlas of physiology,, 6th Edition ed., Thieme Medical Publishers, 2009.

[21] F. N. WILSON, A. G. MACLEOD e P. S. BARKER, "The potential variations produced by the heart beat at the apices of Einthoven's triangle," *American Heart Journal,* vol. 7 No 2, pp. 207-211, 1931.

[22] F. N. WILSON, F. D. JOHNSTON e A. G. MACLEOD, "Electrocardiograms that represent the potential variations of a single electrode," *American Heart Journal,* vol. 9, pp. 447-471, 1934.

[23] J. MALMIVUO e R. PLONSEY, Bioelectromagnetism: principles and applications of bioelectric and biomagnetic fields, Oxford University Press, 1995.

[24] R. MASON e I. LIKAR, "A new system of multiple-lead exercise electrocardiography," *American Heart Journal,* vol. 71 No. 2, pp. 196-205, 1966.

[25] D. B. GESELOWITZ, "Dipole theory in electrocardiography," *The American Journal of Cardiology,* vol. 14, pp. 301-306, 1964.

[26] B. M. NIGG e W. HERZOG, Biomechanics of the musculo-skeletal system, Jon Wiley and Sons, 1999.

[27] J. G. WEBSTER, "Medical Instrumentation: Application and Design", Fourth Edition, 2010.

[28] G. SCHALK and J. MELLINGER, "A Practical Guide to BrainComputer Interfacing with BCI2000," London: Springer-Verlag, 2010.

[29] J. R. WOLPAW, N. BIRBAUME, D. J. McFARLAND, G. PFURTSCHELLER and T. M.VAUGHAN, Braincomputer interfaces for communication and control, in Clin. Neurophysiol. , vol. 113, pp. 76791, 2001.

[30] G. SCHALK, D. J. McFARLAND, T. HINTERBERGER, N. BIRBAUMER and J. R. WOLPAW, BCI2000: a general-purpose brain-computer interface (BCI) system, in IEEE Trans. Biomed. Eng., vol. 51, pp. 10341043, 2004.

[31] M. M. JACKSON and R. MAPPUS, D. S. Tan "Brain-Computer Interfaces: Applying Our Minds to Human-Computer Interaction, Chapter 6, Applications for Brain-Computer Interfaces"

[32] "*g.tec MOBIlab+ Device*".
Disponível em: http://www.gtec.at/Products/Hardware-and-Accessories/g.MOBIlab-Specs-Features
Consultado em 14 de Julho de 2015.

[33] "*Enobio NE Device"*.
Disponível em: http://www.neuroelectrics.com/products/enobio/enobio-8/#teach_spects
Consultado em 29 de Setembro de 2015.

[34] "*Modular EEG*".
Disponível em: http://openeeg.sourceforge.net/doc/
Consultado em 14 de Julho de 2015.

[35] *"PLUX BioSignals Device"*.
Disponível em: http://biosignalsplux.com/index.php/en/
Consultado em 14 de Julho de 2015.

[36] *"BITalino BioSignals Device"*.
Disponível em: http://bitalino.com/
Consultado em 14 de Julho de 2015.

[37] *"Data Sheet ADS1299"*.
Disponível em: http://www.ti.com/lit/ds/symlink/ads1299.pdf
Consultado em 05 Setembro de 2014.

[38] *"Manual ADS1299 AFE + DSP MMB0"*.
Disponível em: http://www.ti.com/lit/ug/slau443a/slau443a.pdf
Consultado em 05 Setembro de 2014.

[39] "ARDUINO UNO".
Disponível em: http://comphaus.com.br/home/wp-content/uploads/2014/03/Arduino-Uno-R3-Front.jpg
Consultado em 05 Setembro de 2014.

[40] *"Bluetooth* HC-05 *Module"*.
Disponível em: https://developer.mbed.org/media/uploads/edodm85/1-881-.jpg
Consultado em 05 Setembro de 2014.

Anexos

Nas próximas secções serão apresentados os anexos que foram referidos em cada um dos capítulos deste trabalho.

Anexo A – Configuração dos Registos Internos e dos *Jumpers* do Protótipo I e II

A configuração dos registos internos e dos *jumpers* do protótipo I e II, vai depender da função e objetivos pretendidos pelo utilizador, sendo que na Tabela A.1 e na Figura A.1 podem ser visualizadas estas configurações.

Tabela A.1 – Configuração *Default* dos registos internos do protótipo I e II.

Registos	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID	0	0	1	1	1	1	1	0
CONGIG1	1	0	0	1	0	1	1	0
CONFIG2	1	1	0	0	0	0	0	0
CONFIG3	0	1	1	0	0	0	0	0
LOFF	0	0	1	0	0	0	0	0
CH1SET	0	1	1	0	0	0	0	0
CH2SET	0	1	1	0	0	0	0	0
CH3SET	0	1	1	0	0	0	0	0
CH4SET	0	1	1	0	0	0	0	0
CH5SET	0	1	1	0	0	0	0	0
CH76SET	0	1	1	0	0	0	0	0
CH7SET	0	1	1	0	0	0	0	0
CH8SET	0	1	1	0	0	0	0	0
BIASSENSP	0	0	0	0	0	0	0	0
BIASSENSN	0	0	0	0	0	0	0	0
LOFF_SENSP	0	0	0	0	0	0	0	0
LOFF_SENSN	0	0	0	0	0	0	0	0
LOFF_FLIP	0	0	0	0	0	0	0	0
LOFF_STATP	0	0	0	0	0	0	0	0
LOFF_STATN	0	0	0	0	0	0	0	0
GPIO	0	0	0	0	0	0	0	0
MISC1	0	0	0	0	0	0	0	0
MISC2	0	0	0	0	0	0	0	0
CONFIG4	0	0	0	0	0	0	0	0

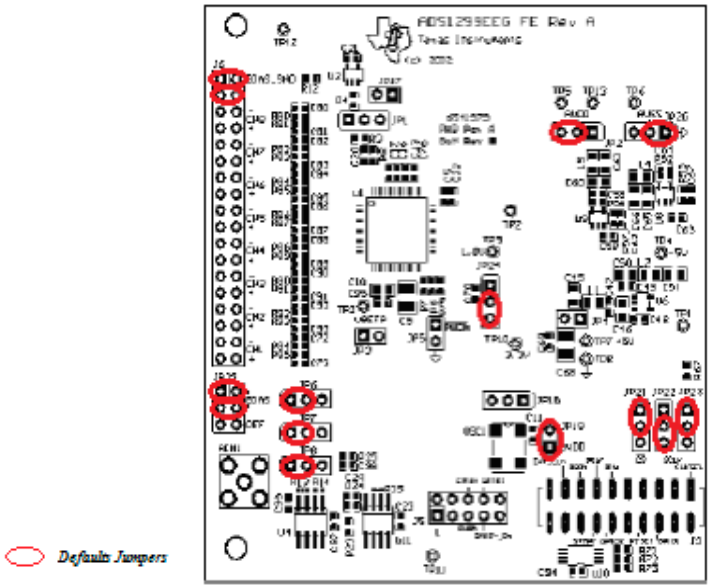


Figura A.1 – Configuração *Default* dos *Jumpers* do protótipo I e II.

Anexo B – Configuração dos Registos Internos e dos *Jumpers* para a Medição do Nível do Ruído do Protótipo I e II

A configuração dos registos internos e dos *jumpers* que permitem a medição do nível do ruído no Protótipo I e II são apresentadas na Tabela B.1 e na Figura B.1. De referir que os registos modificados são apresentados de cor vermelha.

Tabela B.1 – Configuração dos registos internos para medição do nível de ruído do protótipo I e II.

Registos	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID	0	0	1	1	1	1	1	0
CONGIG1	1	0	0	1	0	1	1	0
CONFIG2	1	1	0	0	0	0	0	0
CONFIG3	0	1	1	0	0	0	0	0
LOFF	0	0	0	0	0	0	0	0
CH1SET	0	1	1	0	0	0	0	**1**
CH2SET	0	1	1	0	0	0	0	**1**
CH3SET	0	1	1	0	0	0	0	**1**
CH4SET	0	1	1	0	0	0	0	**1**
CH5SET	0	1	1	0	0	0	0	**1**
CH76SET	0	1	1	0	0	0	0	**1**
CH7SET	0	1	1	0	0	0	0	**1**
CH8SET	0	1	1	0	0	0	0	**1**
BIASSENSP	0	0	0	0	0	0	0	0
BIASSENSN	0	0	0	0	0	0	0	0
LOFF_SENSP	0	0	0	0	0	0	0	0
LOFF_SENSN	0	0	0	0	0	0	0	0
LOFF_FLIP	0	0	0	0	0	0	0	0
LOFF_STATP	0	0	0	0	0	0	0	0
LOFF_STATN	0	0	0	0	0	0	0	0
GPIO	0	0	0	0	0	0	0	0
MISC1	0	0	0	0	0	0	0	0
MISC2	0	0	0	0	0	0	0	0
CONFIG4	0	0	0	0	0	0	0	0

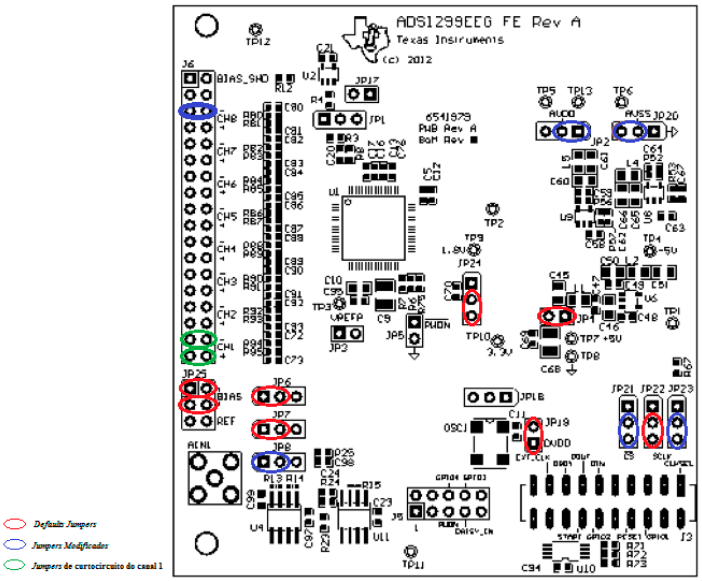


Figura B.1 – Configuração dos *jumpers* para medição do nível de ruído do protótipo I e II.

Anexo C – Configuração dos Registos e dos *Jumpers* para obtenção de ondas geradas internamente pelo ADS1299

A configuração dos registos internos e dos *jumpers* que permitem a obtenção de ondas geradas internamente pelo ADS1299 com protótipo I e II são apresentadas na Tabela C.1 e na Figura C.1. De referir que os registos modificados são apresentados de cor vermelha.

Tabela C.1 – Configuração dos registos internos para a obtenção de geradas internamente pelo ADS1299 com o protótipo I e II.

Registos	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID	0	0	1	1	1	1	1	0
CONGIG1	1	0	0	1	0	1	1	0
CONFIG2	1	1	0	0	0	**X**	**X**	**X**
CONFIG3	**1**	1	1	0	0	0	0	0
LOFF	0	0	0	0	0	0	0	0
CH1SET	0	1	1	0	0	0	0	0
CH2SET	0	1	1	0	0	0	0	0
CH3SET	0	1	1	0	0	0	0	0
CH4SET	0	1	1	0	0	0	0	0
CH5SET	0	1	1	0	0	0	0	0
CH76SET	0	1	1	0	0	0	0	0
CH7SET	0	1	1	0	0	0	0	0
CH8SET	0	1	1	0	0	0	0	0
BIASSENSP	0	0	0	0	0	0	0	0
BIASSENSN	0	0	0	0	0	0	0	0
LOFF_SENSP	0	0	0	0	0	0	0	0
LOFF_SENSN	0	0	0	0	0	0	0	0
LOFF_FLIP	0	0	0	0	0	0	0	0
LOFF_STATP	0	0	0	0	0	0	0	0
LOFF_STATN	0	0	0	0	0	0	0	0
GPIO	0	0	0	0	0	0	0	0
MISC1	0	0	0	0	0	0	0	0
MISC2	0	0	0	0	0	0	0	0
CONFIG4	0	0	0	0	0	0	0	0

Atendendo a Tabela C.1 pode-se visualizar que os bits 0, 1 e 2 do registo CONFIG2 encontra-se com o valor X, isto quer dizer que existem 4 modos de configuração que já foram abordados no capítulo 4, pelo que não serão novamente expostos.

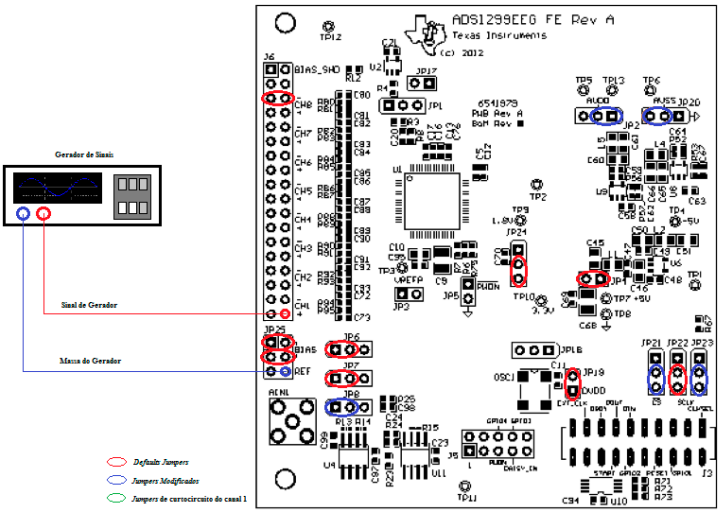


Figura C.1– Configuração dos *jumpers* para a obtenção de geradas internamente pelo ADS1299 com o protótipo I e II.

Anexo D – Configuração dos Registos e dos *Jumpers* para obtenção de com o protótipo I e II através de um gerador de sinais

A configuração dos registos internos e dos *jumpers* que permitem a obtenção de ondas com protótipo I e II através de um gerador de sinais, são apresentadas na Tabela D.1 e na Figura D.1. De referir que os registos modificados são apresentados de cor vermelha.

Tabela D.1 – Configuração dos registos internos para a obtenção de ondas com o protótipo I e II geradas através de um gerador de sinais.

Registos	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID	0	0	1	1	1	1	1	0
CONGIG1	1	0	0	1	0	1	1	0
CONFIG2	1	1	0	**0**	0	0	0	0
CONFIG3	**1**	1	1	0	0	0	0	0
LOFF	0	0	**1**	0	0	0	0	0
CH1SET	0	1	1	0	0	0	0	0
CH2SET	0	1	1	0	0	0	0	0
CH3SET	0	1	1	0	0	0	0	0
CH4SET	0	1	1	0	0	0	0	0
CH5SET	0	1	1	0	0	0	0	0
CH76SET	0	1	1	0	0	0	0	0
CH7SET	0	1	1	0	0	0	0	0
CH8SET	0	1	1	0	0	0	0	0
BIASSENSP	0	0	0	0	0	0	0	0
BIASSENSN	0	0	0	0	0	0	0	0
LOFF_SENSP	0	0	0	0	0	0	0	0
LOFF_SENSN	0	0	0	0	0	0	0	0
LOFF_FLIP	0	0	0	0	0	0	0	0
LOFF_STATP	0	0	0	0	0	0	0	0
LOFF_STATN	0	0	0	0	0	0	0	0
GPIO	0	0	**1**	0	0	0	0	0
MISC1	0	0	0	0	0	0	0	0
MISC2	0	0	0	0	0	0	0	0
CONFIG4	0	0	0	0	0	0	0	0

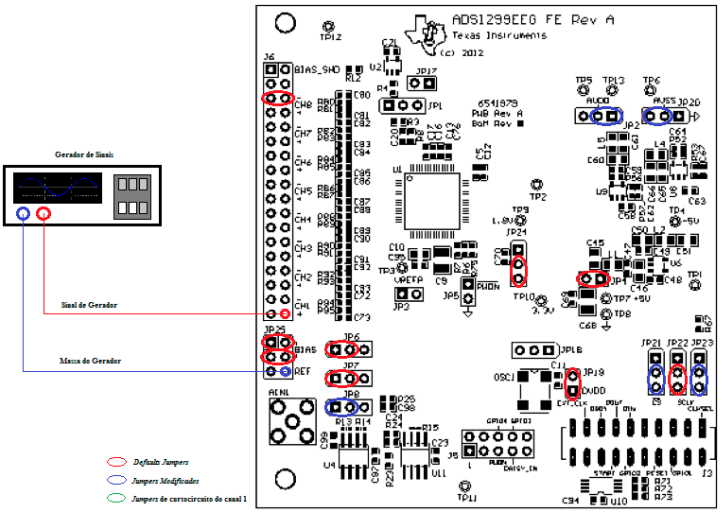


Figura D.1 – Configuração dos *jumpers* para a obtenção de ondas com o protótipo I e II geradas através de um gerador de sinais.

Anexo E – Código para o funcionamento do módulo *Bluetooth* no Protótipo II

O código que permite a comunicação *Bluetooth* entre o protótipo II e um PC é apresentado de seguida:

```
/*-------------------------------------------------------------------------
-------------------------------------------------------------
* File Name: BluetoothProgramArd.ino
* Author: Chris Rorden based on Arduino code by Ryan Hunt
* Created in: 2013
* Modified: 2014 by Miguel Sousa
-------------------------------------------------------------------------
-------------------------------------------------------------*/

     char*  pin =          "0000";                    // Pairing Code for
Module, 4 digits only.. (0000-9999)
     #if defined(__SAM3X8E__) //if Arduino Due
       #define WiredSerial SerialUSB //Due Native port
       const long bps = 115200;//desired bps
       char* name =    "us115k0000bt";//"us922k0000bt"; //any name you want
-Teensy/Arduino names start "usbmodem", so  "us" can help detect if Teensy
is attached, you could also add pin number (0000)
       int  led =             13;   // Pin of Blinking LED, pin 13 for
Arduino/Teensy3, Pin 11 for Teensy2
       //For Due:  Connect BtRx to Tx1 (pin18) and BtTx to Rx1 (pin 19)
     #elif defined(__MK20DX128__) //if Teensy 3.0...
       #define WiredSerial Serial
       const long bps =  460800;//desired bps
       char* name =    "us461k0000bt";//"us922k0000bt"; //any name you want
-Teensy/Arduino names start "usbmodem", so  "us" can help detect if Teensy
is attached, you could also add pin number (0000)
       int  led =             13;   // Pin of Blinking LED, pin 13 for
Arduino/Teensy3, Pin 11 for Teensy2
       #define Hardserial
       //For Teensy 3: Connect BtRx to Tx1 (pin1) and BtTx to Rx1 (pin 0)
     #elif defined(__AVR_ATmega32U4__) && defined(CORE_TEENSY)  //if Teensy
2.0...
       #define WiredSerial Serial
       const long bps = 115200;//desired bps
       char* name =    "us115k0000bt";//"us922k0000bt"; //any name you want
-Teensy/Arduino names start "usbmodem", so  "us" can help detect if Teensy
is attached, you could also add pin number (0000)
       int  led =             11;    // Pin of Blinking LED, pin 13 for
Arduino/Teensy3, Pin 11 for Teensy2
       //For Teensy2:  Connect BtRx to Tx (pin8) and BtTx to Rx (pin 7)
     #else //assume AVR devices like a Leonardo
       #define WiredSerial Serial
       const long bps = 19200; //desired BPS
       char* name =     "us19k0000bt";//any name you want -Teensy/Arduino
names start "usbmodem", so  "us" can help detect if Teensy is attached, you
could also add pin number (0000)
       int  led =             13;   // Pin of Blinking LED, pin 13 for
Arduino/Teensy3, Pin 11 for Teensy2
       //For Leonardo:  Connect BtRx to Tx1 (pin1) and BtTx to Rx1 (pin 0)
     #endif

     #define  bt  Serial1  //Bluetooth  module  attached  to  Serial1  (pins
TX1/RX1)
```

```
    //HardwareSerial bt = HardwareSerial(); //If you get an error here,
uncomment line #define Hardserial
    //#include <SoftwareSerial.h> //If you get an error here, comment this
line
    //SoftwareSerial    bt(0, 1);  //If you get an error here, comment out
line #define Hardserial and uncomment "#include <SoftwareSerial.h>"

     const int numPossibleBps = 12;
     long possibleBps[numPossibleBps] = {1200, 2400, 4800, 9600, 19200,
38400, 57600,115200, 230400, 460800, 921600, 1382400};
     long origBpsIndex = 3; //original BPS - typically 3rd index = 9600
bps
     int wait = 1000;
     void setup()
    {
      pinMode(led, OUTPUT);
      digitalWrite(led, LOW); // Turn off LED to signal waiting for
Terminal
      delay(1000); // Catch Due reset problem
      WiredSerial.begin(9600); //use native port on Due
      while (WiredSerial.read() >= 0) {}
//http://forum.arduino.cc/index.php?topic=134847.0
      delay(200); // Catch Due reset probleme
      while (!WiredSerial) ; //Teensy2 users may need to comment this line
- required by Leonardo http://arduino.cc/en/Serial/IfSerial (ads129n
requires 3.3v signals, Leonardo is 5v)
      digitalWrite(led, HIGH);                    // Turn on LED to signal
programming has started
      WiredSerial.println("Configuring bluetooth module for use with
OpenEMG, please wait.");
      //attempt connection
      int index = origBpsIndex;
      bt.begin(possibleBps[index]);                              // Speed of
your bluetooth module, 9600 is default from factory.
      delay(wait);
      bt.print("AT");
      delay(wait);
      //test for good connection
      int  resp =  bt.read();
      if (resp == -1) index = 0;
      while ((resp == -1) && (index < numPossibleBps)) {
         bt.begin(possibleBps[index]);// Speed of your bluetooth module,
9600 is default from factory.
         WiredSerial.print("Attempting to connect at ");
WiredSerial.print(possibleBps[index]); WiredSerial.println("bps");
         delay(wait);
         bt.print("AT");
         delay(wait);
         resp = bt.read();
         if (resp == -1) index ++;
      } //while no response
      if (resp == -1) {
        WiredSerial.println("ERROR: no response from Bluetooth device.
Please check connections and origbps ");
        digitalWrite(led, LOW);
      } else {
         WiredSerial.print("Successfully connected at ");
WiredSerial.print(possibleBps[index]); WiredSerial.println("bps");
      }
      bt.print("AT+VERSION");
        delay(wait);
```

```
        WiredSerial.print("Setting PIN : ");WiredSerial.println(pin);
        bt.print("AT+PIN"); bt.print(pin);
          delay(wait);
        WiredSerial.print("Setting NAME: "); WiredSerial.println(name);
        bt.print("AT+NAME");
        bt.print(name);
          delay(wait);
        WiredSerial.print("Setting   BAUD:   ");   WiredSerial.println(bps);
//Report baud rate
        switch (bps) {
          case 1200:
            bt.print("AT+BAUD1");
            break;
          case 2400:
            bt.print("AT+BAUD2");
            break;
          case 4800:
            bt.print("AT+BAUD3");
            break;
          case 9600:
            bt.print("AT+BAUD4");
            break;
          case 19200:
            bt.print("AT+BAUD5");
            break;
          case 38400:
            bt.print("AT+BAUD6");
            break;
          case 57600:
            bt.print("AT+BAUD7");
            break;
          case 115200:
            bt.print("AT+BAUD8");
            break;
          case 230400:
             bt.print("AT+BAUD9");
             break;
          case 460800:
            bt.print("AT+BAUDA");
            break;
          case 921600:
            bt.print("AT+BAUDB");
            break;
          case 1382400:
            bt.print("AT+BAUDC");
            break;
          default:
            bt.print("AT+BAUD4"); //9600
            WiredSerial.println("WARNING: Unknown baud rate - setting to
default 9600bps");
        }
        delay(wait);
        WiredSerial.println("Hopefully bluetooth device is reset");
      }
      void loop() {
        // nothing to do, its all in the interrupt handler!
      }
```

Anexo F – Modificação do módulo do Arduino UNO

A construção do Protótipo II envolveu a modificação do módulo do Arduino UNO, nomeadamente na sua alimentação, a qual passo a funcionar a 3,3 V em vez de 5 V. Os passos realizados para a referida modificação são expostos de seguida:

Primeiro Passo: Remover o regulador de tensão de 5 V (**Error! Reference source not found.**).

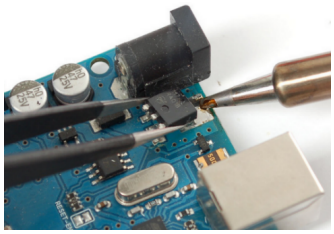


Figura F.1 – Remoção do Regulador de Tensão de 5 V do Arduino UNO.

Segundo Passo: No lugar do regulador de tensão retirado, foi colocado um regulador de tensão de 3,3 V (Figura F.2).

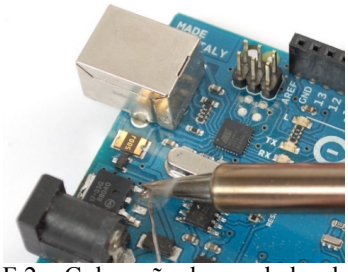
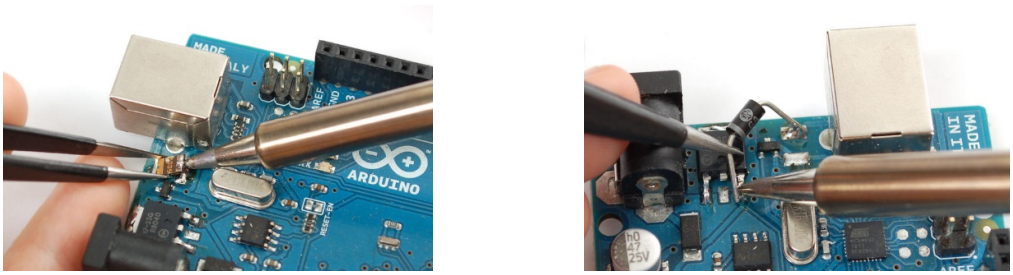


Figura F.2 – Colocação do regulador de tensão 3,3 V.

Terceiro Passo: Apesar de parecer que o módulo já está pronto, foi necessário remover o fusível de proteção do módulo, pois a tomada USB fornece 5 V e está ligada diretamente à saída do regulador de tensão (Figura F.3-a). O fusível foi substituído por um díodo de potência, 1N4001 (Figura F.3-b). Embora não ter o fusível na tomada USB possa parecer uma opção insegura, os computadores normalmente têm seus próprios fusíveis no conector USB, garantindo a mesma segurança.



(a) (b)

Figura F.3 – Montagem de Componentes: (a) Remoção do fusível do módulo Arduino UNO; e (b) Colocação do díodo no módulo Arduino UNO.

Feitas as alterações mencionadas anteriormente o Arduino UNO passa a operar a 3,3 V e pode por isso ser conectado diretamente ao dispositivo ADS1299.

O módulo *Bluetooth* escolhido opera a 3,3 V, pelo que após a alteração descrita anteriormente pode ser ligado diretamente ao Arduino sem qualquer problema. Caso não tivesse sido alterada a tensão de operação do Arduino seria necessário de utilizar divisores de tensão para não danificar o dispositivo *Bluetooth*.

Anexo G – Configuração de Entradas Unipolares no Protótipo II

Para o colocar as entradas do protótipo II em modo unipolar, é necessário configurar os seus registos internos, sendo estes apresentados na Tabela G.1. De referir que os registos modificados são apresentados de cor vermelha.

Tabela G.1 – Configuração dos registos internos do protótipo II para o funcionamento de entradas unipolares.

Registos	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID	0	0	1	1	1	1	1	0
CONGIG1	1	0	0	1	0	1	1	0
CONFIG2	1	1	0	**0**	0	0	0	0
CONFIG3	**1**	1	1	0	0	0	0	0
LOFF	0	0	**1**	0	0	0	0	0
CH1SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH2SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH3SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH4SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH5SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH76SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH7SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH8SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
BIASSENSP	0	0	0	0	0	0	0	0
BIASSENSN	0	0	0	0	0	0	0	0
LOFF_SENSP	0	0	0	0	0	0	0	0
LOFF_SENSN	0	0	0	0	0	0	0	0
LOFF_FLIP	0	0	0	0	0	0	0	0
LOFF_STATP	0	0	0	0	0	0	0	0
LOFF_STATN	0	0	0	0	0	0	0	0
GPIO	0	0	0	0	0	0	0	0
MISC1	0	0	**1**	0	0	0	0	0
MISC2	0	0	0	0	0	0	0	0
CONFIG4	0	0	0	0	0	0	0	0

Para além da configuração de registos ainda é necessário modificar alguns dos seus *jumpers* e colocar os elétrodos nas posições indicadas na Figura G.1.

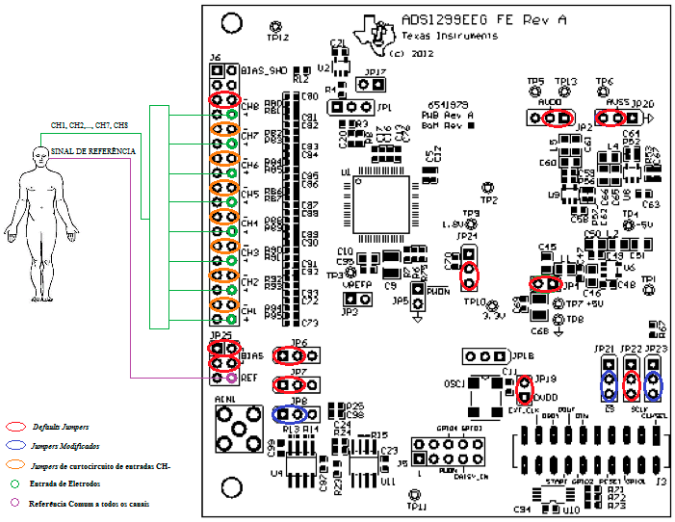


Figura G.1 – Configuração de entradas unipolares no Protótipo II.

Anexo H – Configuração de Entradas Diferenciais no Protótipo II

Para o colocar as entradas do protótipo II em modo diferencial, é necessário configurar os seus registos internos, sendo estes apresentados na Tabela H.1. De referir que os registos modificados são apresentados de cor vermelha.

Tabela H.1 – Configuração dos registos internos do protótipo II para o funcionamento de entradas diferenciais.

Registos	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID	0	0	1	1	1	1	1	0
CONGIG1	1	0	0	1	0	1	1	0
CONFIG2	1	1	0	**0**	0	0	0	0
CONFIG3	**1**	1	1	0	0	0	0	0
LOFF	0	0	**1**	0	0	0	0	0
CH1SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH2SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH3SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH4SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH5SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH76SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH7SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
CH8SET	0	**1**	**1**	**0**	0	**0**	**0**	**0**
BIASSENSP	0	0	0	0	0	0	0	0
BIASSENSN	0	0	0	0	0	0	0	0
LOFF_SENSP	0	0	0	0	0	0	0	0
LOFF_SENSN	0	0	0	0	0	0	0	0
LOFF_FLIP	0	0	0	0	0	0	0	0
LOFF_STATP	0	0	0	0	0	0	0	0
LOFF_STATN	0	0	0	0	0	0	0	0
GPIO	0	0	0	0	0	0	0	0
MISC1	0	0	0	0	0	0	0	0
MISC2	0	0	0	0	0	0	0	0
CONFIG4	0	0	0	0	0	0	0	0

Para além da configuração de registos ainda é necessário modificar alguns dos seus *jumpers* e colocar os elétrodos nas posições indicadas na Figura G.1.

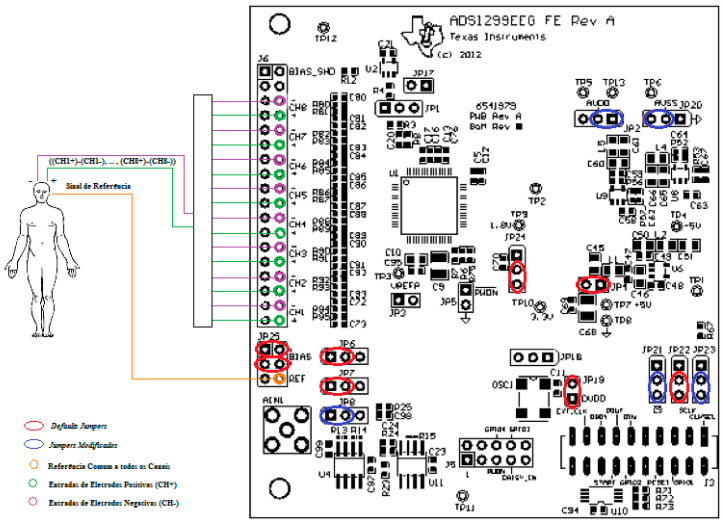


Figura H.1 – Configuração de entradas diferenciais no Protótipo II.

Anexo I – Código do Firmware do Arduíno UNO

Neste anexo será apresentado todo o firmware do Arduino UNO, o qual encontra-se organizado da seguinte maneira:

Pasta Principal (*Main*): Ficheiro "StreamRawData.ino"

```
/*-------------------------------------------------------------------------
-------------------------------------------------------------
* File Name: StreamRawData.ino
* Author: Chip Audette, Joel Murphy and Connor Russomanno
* Created in: 2013
* Modified: 2014 by Miguel Sousa
-------------------------------------------------------------------------
-------------------------------------------------------------*/
     typedef long int int32;

     #define N_CHANNELS_PER_OPENBCI (8) //number of channels on a single
OpenBCI board

     //for using a single OpenBCI board
     #include <ADS1299Manager.h>  //for a single OpenBCI board
     ADS1299Manager ADSManager;   //Uses SPI bus and pins to say data is
ready.  Uses Pins 13,12,11,10,9,8,4
     #define MAX_N_CHANNELS (N_CHANNELS_PER_OPENBCI)//how many channels are
available in hardware
     //#define  MAX_N_CHANNELS  (2*N_CHANNELS_PER_OPENBCI)      //how  many
channels are available in hardware...use this for daisy-chained board
     int nActiveChannels = MAX_N_CHANNELS;     //how many active channels
would I like?

     //other settings for OpenBCI
     byte gainCode = ADS_GAIN24;   //how much gain do I want
     byte inputType = ADSINPUT_NORMAL;    //here's the normal way to setup
the channels
     //byte inputType = ADSINPUT_SHORTED;   //here's another way to setup
the channels
     //byte inputType = ADSINPUT_TESTSIG;   //here's a third way to setup
the channels

     //other variables
     long sampleCounter = 0;      //used to time the tesing loop
     boolean is_running = false; //this flag is set in serialEvent on
reciept of prompt
     #define PIN_STARTBINARY (7)//pull this pin to ground to start binary
transfer
     //define PIN_STARTBINARY_OPENEEG (6)
     boolean startBecauseOfPin = false;
     boolean startBecauseOfSerial = false;

     //analog input
     #define PIN_ANALOGINPUT (A0)
     int analogVal = 0;

     #define OUTPUT_NOTHING (0)
     #define OUTPUT_TEXT (1)
     #define OUTPUT_BINARY (2)
     #define OUTPUT_BINARY_SYNTHETIC (3)
     #define OUTPUT_BINARY_4CHAN (4)
```

```
    #define OUTPUT_BINARY_OPENEEG (6)
    #define OUTPUT_BINARY_OPENEEG_SYNTHETIC (7)
    #define OUTPUT_BINARY_WITH_AUX (8)
    int outputType;

    //Design filters  (This BIQUAD class requires ~6K of program space! Ouch.)
    //For frequency response of these filters: http://www.earlevel.com/main/2010/12/20/biquad-calculator/
    #include <Biquad_multiChan.h>    //modified from this source code: http://www.earlevel.com/main/2012/11/26/biquad-c-source-code/
    #define SAMPLE_RATE_HZ (250.0)  //default setting for OpenBCI
    #define FILTER_Q (0.5)               //critically damped is 0.707 (Butterworth)
    #define FILTER_PEAK_GAIN_DB (0.0) //we don't want any gain in the passband
    #define HP_CUTOFF_HZ (0.5)  //set the desired cutoff for the highpass filter
    Biquad_multiChan stopDC_filter(MAX_N_CHANNELS,bq_type_highpass,HP_CUTOFF_HZ / SAMPLE_RATE_HZ, FILTER_Q, FILTER_PEAK_GAIN_DB); //one for each channel because the object maintains the filter states
    //Biquad_multiChan  stopDC_filter(MAX_N_CHANNELS,bq_type_bandpass,10.0 / SAMPLE_RATE_HZ, 6.0, FILTER_PEAK_GAIN_DB); //one for each channel because the object maintains the filter states
    #define NOTCH_FREQ_HZ (60.0)
    #define NOTCH_Q (4.0)              //pretty sharp notch
    #define NOTCH_PEAK_GAIN_DB (0.0)   //doesn't matter for this filter type
    Biquad_multiChan notch_filter1(MAX_N_CHANNELS,bq_type_notch,NOTCH_FREQ_HZ / SAMPLE_RATE_HZ, NOTCH_Q, NOTCH_PEAK_GAIN_DB); //one for each channel because the object maintains the filter states
    Biquad_multiChan notch_filter2(MAX_N_CHANNELS,bq_type_notch,NOTCH_FREQ_HZ / SAMPLE_RATE_HZ, NOTCH_Q, NOTCH_PEAK_GAIN_DB); //one for each channel because the object maintains the filter states
    boolean useFilters = false;  //enable or disable as you'd like...turn off if you're daisy chaining!


    void setup() {
      //detect which version of OpenBCI we're using (is Pin2 jumped to Pin3?)
      int OpenBCI_version = OPENBCI_V2;  //assume V2
      pinMode(2,INPUT);    digitalWrite(2,HIGH);  //activate pullup...for detecting which version of OpenBCI PCB
      pinMode(3,OUTPUT); digitalWrite(3,LOW);  //act as a ground pin...for detecting which version of OpenBCI PCB
      if (digitalRead(2) == LOW) OpenBCI_version = OPENBCI_V1; //check pins to see if there is a jumper.  if so, it is the older board
      boolean isDaisy = false; if (MAX_N_CHANNELS > 8) isDaisy = true;
      ADSManager.initialize(OpenBCI_version,isDaisy);  //must do this VERY early in the setup...preferably first

      // setup the serial link to the PC
      if (MAX_N_CHANNELS > 8) {
        Serial.begin(115200*2);  //Need 115200 for 16-channels, only need 115200 for 8-channels but let's do 115200*2 for consistency
      } else {
        Serial.begin(115200);
```

```
      }
      Serial.println(F("ADS1299-Arduino UNO - Stream Raw Data")); //read
the string from Flash to save RAM
      Serial.print(F("Configured as OpenBCI_Version code =
"));Serial.print(OpenBCI_version);  Serial.print(F(", isDaisy =
"));Serial.println(ADSManager.isDaisy);
      Serial.print(F("Configured for "));Serial.print(MAX_N_CHANNELS);
Serial.println(F(" Channels"));
      Serial.flush();

      // setup the channels as desired on the ADS1299..set gain, input
type, referece (SRB1), and patient bias signal
      for (int chan=1; chan <= nActiveChannels; chan++) {
        ADSManager.activateChannel(chan, gainCode, inputType);
      }

      //setup the lead-off detection parameters
      ADSManager.configureLeadOffDetection(LOFF_MAG_6NA,
LOFF_FREQ_31p2HZ);

      //print state of all registers
      ADSManager.printAllRegisters();Serial.flush();

      // setup hardware to allow a jumper or button to start the
digitaltransfer
      pinMode(PIN_STARTBINARY,INPUT);  digitalWrite(PIN_STARTBINARY,HIGH);
//activate pullup
      //pinMode(PIN_STARTBINARY_OPENEEG,INPUT);
digitalWrite(PIN_STARTBINARY_OPENEEG,HIGH);  //activate pullup

      //look out for daisy chaining and disable filtering because it'll
likely take too much computation
      if (nActiveChannels > 8) useFilters = false;
      if (useFilters) Serial.print(F("Configured to do some filtering here
on the Arduino."));


      // tell the controlling program that we're ready to start!
      Serial.println(F("Press '?' to query and print ADS1299 register
settings again")); //read it straight from flash
      Serial.println(F("Press 1-8 to disable EEG Channels, q-i to enable
(all enabled by default)"));
      Serial.println(F("Press 'f' to enable filters.  'g' to disable
filters"));
      Serial.println(F("Press 'x' (text) or 'b' (binary) to begin
streaming data..."));

    } // end of setup

    boolean firstReport = true;
    unsigned long totalMicrosBusy = 0;  //use this to count time
    void loop(){

      if (digitalRead(PIN_STARTBINARY)==LOW) {
        //button is pressed (or pin is jumpered to ground)
        startBecauseOfPin = true;
        //startRunning(OUTPUT_BINARY_OPENEEG_SYNTHETIC);
        startRunning(OUTPUT_BINARY_OPENEEG);
        //if (firstReport) { Serial.println(F("Starting Binary_OpenEEG
Based on Pin")); firstReport=false;}
      } else {
```

```
        if (startBecauseOfPin) {
          startBecauseOfPin = false;
          stopRunning();
          //if (firstReport == false) { Serial.println(F("Stopping Binary
Based on Pin")); firstReport=true;}
        }
      }

      if (is_running) {

        //is data ready?
        while(!(ADSManager.isDataAvailable())){                // watch the
DRDY pin
          delayMicroseconds(100);
        }
        unsigned long start_micros = micros();

        //get the data
        analogVal = analogRead(PIN_ANALOGINPUT);   // get analog value
        ADSManager.updateChannelData();                       //  update  the
channelData array
        sampleCounter++;                                 // increment my sample
counter

        //Apply  filers to the data
        if (useFilters) applyFilters();

        //print the data
        switch (outputType) {
          case OUTPUT_NOTHING:
            //don't output anything...the Arduino is still collecting data
from the OpenBCI board...just nothing is being done with it
            //if ((sampleCounter % 250) == 1) { Serial.print(F("Free RAM =
")); Serial.println(freeRam()); }; //print memory status
            break;
          case OUTPUT_BINARY:

ADSManager.writeChannelDataAsBinary(MAX_N_CHANNELS,sampleCounter);   //print
all channels, whether active or not
            break;
          case OUTPUT_BINARY_WITH_AUX:

ADSManager.writeChannelDataAsBinary(MAX_N_CHANNELS,sampleCounter,(long
int)analogVal);  //print all channels, whether active or not
            break;
          case OUTPUT_BINARY_SYNTHETIC:

ADSManager.writeChannelDataAsBinary(MAX_N_CHANNELS,sampleCounter,(boolean)t
rue);  //print all channels, whether active or not
            break;
          case OUTPUT_BINARY_4CHAN:
            ADSManager.writeChannelDataAsBinary(4,sampleCounter);   //print
4 channels, whether active or not
            break;
          case OUTPUT_BINARY_OPENEEG:
            ADSManager.writeChannelDataAsOpenEEG_P2(sampleCounter);
//this format accepts 6 channels, so that's what it does
            break;
          case OUTPUT_BINARY_OPENEEG_SYNTHETIC:
            ADSManager.writeChannelDataAsOpenEEG_P2(sampleCounter,true);
//his format accepts 6 channels, so that's what it does
```

```
            break;
          default:

ADSManager.printChannelDataAsText(MAX_N_CHANNELS,sampleCounter);     //print
all channels, whether active or not
        }


    //    totalMicrosBusy += (micros()-start_micros); //accumulate
    //    if (sampleCounter==250) totalMicrosBusy = 0;  //start from 250th
sample
    //    if (sampleCounter==500) {
    //      stopRunning();
    //      Serial.println();
    //      Serial.print(F("Was busy for "));
    //      Serial.print(totalMicrosBusy);
    //      Serial.println(F(" microseconds across 250 samples"));
    //       Serial.print(F("Assuming a 250Hz Sample Rate, it was busy for
"));
    //      unsigned long micros_per_250samples = 1000000UL;
    //
Serial.print(((float)totalMicrosBusy/(float)micros_per_250samples)*100.0);
    //      Serial.println(F("% of the available time"));
    //    }

      }

    } // end of loop


    #define ACTIVATE_SHORTED (2)
    #define ACTIVATE (1)
    #define DEACTIVATE (0)
    void serialEvent(){              // send an 'x' on the serial line to
trigger ADStest()
      while(Serial.available()){
        char inChar = (char)Serial.read();
        switch (inChar)
        {
          //turn channels on and off
          case '1':
            changeChannelState_maintainRunningState(1,DEACTIVATE); break;
          case '2':
            changeChannelState_maintainRunningState(2,DEACTIVATE); break;
          case '3':
            changeChannelState_maintainRunningState(3,DEACTIVATE); break;
          case '4':
            changeChannelState_maintainRunningState(4,DEACTIVATE); break;
          case '5':
            changeChannelState_maintainRunningState(5,DEACTIVATE); break;
          case '6':
            changeChannelState_maintainRunningState(6,DEACTIVATE); break;
          case '7':
            changeChannelState_maintainRunningState(7,DEACTIVATE); break;
          case '8':
            changeChannelState_maintainRunningState(8,DEACTIVATE); break;
          case 'q':
            changeChannelState_maintainRunningState(1,ACTIVATE); break;
          case 'w':
            changeChannelState_maintainRunningState(2,ACTIVATE); break;
          case 'e':
```

```
            changeChannelState_maintainRunningState(3,ACTIVATE); break;
          case 'r':
            changeChannelState_maintainRunningState(4,ACTIVATE); break;
          case 't':
            changeChannelState_maintainRunningState(5,ACTIVATE); break;
          case 'y':
            changeChannelState_maintainRunningState(6,ACTIVATE); break;
          case 'u':
            changeChannelState_maintainRunningState(7,ACTIVATE); break;
          case 'i':
            changeChannelState_maintainRunningState(8,ACTIVATE); break;

          //turn lead-off detection on and off
          case '!':

changeChannelLeadOffDetection_maintainRunningState(1,ACTIVATE,PCHAN);
break;
          case '@':

changeChannelLeadOffDetection_maintainRunningState(2,ACTIVATE,PCHAN);
break;
          case '#':

changeChannelLeadOffDetection_maintainRunningState(3,ACTIVATE,PCHAN);
break;
          case '$':

changeChannelLeadOffDetection_maintainRunningState(4,ACTIVATE,PCHAN);
break;
          case '%':

changeChannelLeadOffDetection_maintainRunningState(5,ACTIVATE,PCHAN);
break;
          case '^':

changeChannelLeadOffDetection_maintainRunningState(6,ACTIVATE,PCHAN);
break;
          case '&':

changeChannelLeadOffDetection_maintainRunningState(7,ACTIVATE,PCHAN);
break;
          case '*':

changeChannelLeadOffDetection_maintainRunningState(8,ACTIVATE,PCHAN);
break;
          case 'Q':

changeChannelLeadOffDetection_maintainRunningState(1,DEACTIVATE,PCHAN);
break;
          case 'W':

changeChannelLeadOffDetection_maintainRunningState(2,DEACTIVATE,PCHAN);
break;
          case 'E':

changeChannelLeadOffDetection_maintainRunningState(3,DEACTIVATE,PCHAN);
break;
          case 'R':

changeChannelLeadOffDetection_maintainRunningState(4,DEACTIVATE,PCHAN);
break;
```

```
            case 'T':

changeChannelLeadOffDetection_maintainRunningState(5,DEACTIVATE,PCHAN);
break;
            case 'Y':

changeChannelLeadOffDetection_maintainRunningState(6,DEACTIVATE,PCHAN);
break;
            case 'U':

changeChannelLeadOffDetection_maintainRunningState(7,DEACTIVATE,PCHAN);
break;
            case 'I':

changeChannelLeadOffDetection_maintainRunningState(8,DEACTIVATE,PCHAN);
break;
             case 'A':

changeChannelLeadOffDetection_maintainRunningState(1,ACTIVATE,NCHAN);
break;
            case 'S':

changeChannelLeadOffDetection_maintainRunningState(2,ACTIVATE,NCHAN);
break;
            case 'D':

changeChannelLeadOffDetection_maintainRunningState(3,ACTIVATE,NCHAN);
break;
            case 'F':

changeChannelLeadOffDetection_maintainRunningState(4,ACTIVATE,NCHAN);
break;
            case 'G':

changeChannelLeadOffDetection_maintainRunningState(5,ACTIVATE,NCHAN);
break;
            case 'H':

changeChannelLeadOffDetection_maintainRunningState(6,ACTIVATE,NCHAN);
break;
            case 'J':

changeChannelLeadOffDetection_maintainRunningState(7,ACTIVATE,NCHAN);
break;
            case 'K':

changeChannelLeadOffDetection_maintainRunningState(8,ACTIVATE,NCHAN);
break;
            case 'Z':

changeChannelLeadOffDetection_maintainRunningState(1,DEACTIVATE,NCHAN);
break;
            case 'X':

changeChannelLeadOffDetection_maintainRunningState(2,DEACTIVATE,NCHAN);
break;
            case 'C':

changeChannelLeadOffDetection_maintainRunningState(3,DEACTIVATE,NCHAN);
break;
            case 'V':
```

```
changeChannelLeadOffDetection_maintainRunningState(4,DEACTIVATE,NCHAN);
break;
            case 'B':

changeChannelLeadOffDetection_maintainRunningState(5,DEACTIVATE,NCHAN);
break;
            case 'N':

changeChannelLeadOffDetection_maintainRunningState(6,DEACTIVATE,NCHAN);
break;
            case 'M':

changeChannelLeadOffDetection_maintainRunningState(7,DEACTIVATE,NCHAN);
break;
            case '<':

changeChannelLeadOffDetection_maintainRunningState(8,DEACTIVATE,NCHAN);
break;

            //control the bias generation
            case '`':
              ADSManager.setAutoBiasGeneration(true); break;
            case '~':
              ADSManager.setAutoBiasGeneration(false); break;

            //control test signals
            case '0':

activateAllChannelsToTestCondition(ADSINPUT_SHORTED,ADSTESTSIG_NOCHANGE,ADS
TESTSIG_NOCHANGE); break;
            case '-':

activateAllChannelsToTestCondition(ADSINPUT_TESTSIG,ADSTESTSIG_AMP_1X,ADSTE
STSIG_PULSE_SLOW); break;
            case '+':

activateAllChannelsToTestCondition(ADSINPUT_TESTSIG,ADSTESTSIG_AMP_1X,ADSTE
STSIG_PULSE_FAST); break;
            case '=':
              //repeat the line above...just for human convenience

activateAllChannelsToTestCondition(ADSINPUT_TESTSIG,ADSTESTSIG_AMP_1X,ADSTE
STSIG_PULSE_FAST); break;
            case 'p':

activateAllChannelsToTestCondition(ADSINPUT_TESTSIG,ADSTESTSIG_AMP_2X,ADSTE
STSIG_DCSIG); break;
            case '[':

activateAllChannelsToTestCondition(ADSINPUT_TESTSIG,ADSTESTSIG_AMP_2X,ADSTE
STSIG_PULSE_SLOW); break;
            case ']':

activateAllChannelsToTestCondition(ADSINPUT_TESTSIG,ADSTESTSIG_AMP_2X,ADSTE
STSIG_PULSE_FAST); break;

            //other commands
            case 'n':
              toggleRunState(OUTPUT_BINARY_WITH_AUX);
              startBecauseOfSerial = is_running;
```

```
            if (is_running) Serial.println(F("Arduino: Starting binary
(including AUX value)..."));
            break;
          case 'b':
            toggleRunState(OUTPUT_BINARY);
            //toggleRunState(OUTPUT_BINARY_SYNTHETIC);
            startBecauseOfSerial = is_running;
            if (is_running) Serial.println(F("Arduino: Starting
binary..."));
            break;
          case 'v':
            toggleRunState(OUTPUT_BINARY_4CHAN);
            startBecauseOfSerial = is_running;
            if (is_running) Serial.println(F("Arduino: Starting binary 4-
chan..."));
            break;
          case 's':
            stopRunning();
            startBecauseOfSerial = is_running;
            break;
          case 'x':
            toggleRunState(OUTPUT_TEXT);
            startBecauseOfSerial = is_running;
            if (is_running) Serial.println(F("Arduino: Starting
text..."));
            break;
          case 'f':
            useFilters = true;
            Serial.println(F("Arduino: enabaling filters"));
            break;
          case 'g':
            useFilters = false;
            Serial.println(F("Arduino: disabling filters"));
            break;
          case '?':
            //print state of all registers
            ADSManager.printAllRegisters();
            break;
          default:
            break;
        }
      }
    }

    boolean toggleRunState(int OUT_TYPE)
    {
      if (is_running) {
        return stopRunning();
      } else {
        return startRunning(OUT_TYPE);
      }
    }

    boolean stopRunning(void) {
      ADSManager.stop();                    // stop the data acquisition
      is_running = false;
      return is_running;
    }

    boolean startRunning(int OUT_TYPE) {
        outputType = OUT_TYPE;
```

```
        ADSManager.start();    //start the data acquisition
        is_running = true;
        return is_running;
    }

    int changeChannelState_maintainRunningState(int chan, int start)
    {
      boolean is_running_when_called = is_running;
      int cur_outputType = outputType;

      //must stop running to change channel settings
      stopRunning();
      if (start == true) {
        Serial.print(F("Activating channel "));
        Serial.println(chan);
        ADSManager.activateChannel(chan,gainCode,inputType);
      } else {
        Serial.print(F("Deactivating channel "));
        Serial.println(chan);
        ADSManager.deactivateChannel(chan);
      }

      //restart, if it was running before
      if (is_running_when_called == true) {
        startRunning(cur_outputType);
      }
    }

    int  changeChannelLeadOffDetection_maintainRunningState(int  chan,  int
start, int code_P_N_Both)
    {
      boolean is_running_when_called = is_running;
      int cur_outputType = outputType;

      //must stop running to change channel settings
      stopRunning();
      if (start == true) {
        Serial.print(F("Activating channel "));
        Serial.print(chan);
        Serial.println(F(" Lead-Off Detection"));
        ADSManager.changeChannelLeadOffDetection(chan,ON,code_P_N_Both);
      } else {
        Serial.print(F("Deactivating channel "));
        Serial.print(chan);
        Serial.println(F(" Lead-Off Detection"));
        ADSManager.changeChannelLeadOffDetection(chan,OFF,code_P_N_Both);
      }

      //restart, if it was running before
      if (is_running_when_called == true) {
        startRunning(cur_outputType);
      }
    }


    int    activateAllChannelsToTestCondition(int    testInputCode,    byte
amplitudeCode, byte freqCode)
    {
      boolean is_running_when_called = is_running;
      int cur_outputType = outputType;
```

```
        //set the test signal to the desired state
        ADSManager.configureInternalTestSignal(amplitudeCode,freqCode);

        //must stop running to change channel settings
        stopRunning();

        //loop over all channels to change their state
        for (int Ichan=1; Ichan <= 8; Ichan++) {
          ADSManager.activateChannel(Ichan,gainCode,testInputCode);   //Ichan
must be [1 8]...it does not start counting from zero
        }

        //restart, if it was running before
        if (is_running_when_called == true) {
          startRunning(cur_outputType);
        }
      }

      long int runningAve[MAX_N_CHANNELS];
      int applyFilters(void) {
        //scale factor for these coefficients was 32768 = 2^15
        const static long int a0 = 32360L; //16 bit shift?
        const static long int a1 = -2L*a0;
        const static long int a2 = a0;
        const static long int b1 = -64718L; //this is a shift of 17 bits!
        const static long int b2 = 31955L;
        static long int z1[MAX_N_CHANNELS], z2[MAX_N_CHANNELS];
        long int val_int, val_in_down9, val_out, val_out_down9;
        float val;
        for (int Ichan=0; Ichan < MAX_N_CHANNELS; Ichan++) {
          switch (1) {
            case 1:
              //use BiQuad
              val = (float) ADSManager.channelData[Ichan]; //get the stored
value for this sample
              val = stopDC_filter.process(val,Ichan);     //apply DC-blocking
filter
              break;
            case 2:
              //do fixed point, 1st order running ave
              val_int  =  ADSManager.channelData[Ichan];  //get  the  stored
value for this sample
              //runningAve[Ichan]=(     ((512-1)*(runningAve[Ichan]>>2))     +
(val_int>>2) )>>7;  // fs/0.5Hz = ~512 points..9 bits
              //runningAve[Ichan]=(     ((256-1)*(runningAve[Ichan]>>2))     +
(val_int>>2) )>>6;  // fs/1.0Hz = ~256 points...8 bits
              runningAve[Ichan]=(      ((128-1)*(runningAve[Ichan]>>1))      +
(val_int>>1) )>>6;  // fs/2.0Hz = ~128 points...7 bits
              val = (float)(val_int - runningAve[Ichan]);  //remove the DC
              break;
      //      case 3:
      //        val_in_down9 = ADSManager.channelData[Ichan] >> 9; //get the
stored value for this sample...bring 24-bit value down to 16-bit
      //         val_out = (val_in_down9 * a0   + (z1[Ichan]>>9)) >> (16-9);
//8bits were already removed...results in 24-bit value
      //          val_out_down9 = val_out >> 9;   //remove eight bits to go
from 24-bit down to 16 bit
      //          z1[Ichan] = (val_in_down9 * a1 + (z2[Ichan] >> 9) - b1 *
val_out_down9   ) >> (16-9);   //8-bits were pre-removed..end in 24 bit
number
```

```
    //          z2[Ichan] = (val_in_down9 * a2   - b2 * val_out_down9) >>
(16-9); //8-bits were pre-removed...end in 24-bit number
    //        val = (float)val_out;
    //        break;
        }
        val = notch_filter1.process(val,Ichan);        //apply 60Hz notch
filter
        val = notch_filter2.process(val,Ichan);     //apply it again
        ADSManager.channelData[Ichan] = (long) val;  //save the value back
into the main data-holding object
      }
      return 0;
    }

    int freeRam()
    {
      extern int __heap_start, *__brkval;
      int v;
      return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int)
__brkval);
    }
```

Pasta ADS1299: Ficheiro “ADS1299Manager.cpp”

```
/*-------------------------------------------------------------------------
-----------------------------------------------------------------
* File Name: ADS1299Manager.cpp
* Author: Chip Audette, Joel Murphy and Conor Russomanno
* Created in: 2013
* Modified 2014 by Miguel Sousa
-------------------------------------------------------------------------
-----------------------------------------------------------------*/

    #include <ADS1299Manager.h>

    typedef long int32;
    //typedef byte uint8_t;

    void ADS1299Manager::initialize(void) {
         boolean isDaisy = false;
        initialize(OPENBCI_V2,isDaisy);
    }

    //Initilize the ADS1299 controller...call this once
    void ADS1299Manager::initialize(const int version,boolean isDaisy)
    {
      ADS1299::initialize(PIN_DRDY,PIN_RST,PIN_CS,SCK_MHZ,isDaisy);      //
(DRDY pin, RST pin, CS pin, SCK frequency in MHz);
      delay(100);

      verbose = false;       // when verbose is true, there will be Serial
feedback
      setVersionOpenBCI(version);
      reset();

      n_chan_all_boards = OPENBCI_NCHAN_PER_BOARD;
      if (isDaisy) n_chan_all_boards = 2*OPENBCI_NCHAN_PER_BOARD;

      //set default state for internal test signal
```

```
      //ADS1299::WREG(CONFIG2,0b11010000);delay(1);    //set internal test
signal, default amplitude, default speed, datasheet PDF Page 41
      //ADS1299::WREG(CONFIG2,0b11010001);delay(1);    //set internal test
signal, default amplitude, 2x speed, datasheet PDF Page 41

configureInternalTestSignal(ADSTESTSIG_AMP_1X,ADSTESTSIG_PULSE_FAST); //set
internal test signal, default amplitude, 2x speed, datasheet PDF Page 41

      //set default state for lead off detection
      configureLeadOffDetection(LOFF_MAG_6NA,LOFF_FREQ_31p2HZ);
    };

    //set which version of OpenBCI we're using.  This affects whether we
use the
    //positive or negative inputs.  It affects whether we use SRB1 or SRB2
for the
    //referenece signal.  Finally, it affects whether the lead_off signals
are
    //flipped or not.
    void ADS1299Manager::setVersionOpenBCI(const int version)
    {
      if (version == OPENBCI_V1) {
           //set whether to use positive or negative inputs
           use_neg_inputs = false;

           //set SRB2
           for (int i=0; i < OPENBCI_NCHAN_PER_BOARD; i++) {
                use_SRB2[i] = false;
           }
      } else {
           //set whether to use positive or negative inputs
           use_neg_inputs = true;

           //set SRB
           for (int i=0; i < OPENBCI_NCHAN_PER_BOARD; i++) {
                use_SRB2[i] = true;
           }

      }

      //set  whether  or  not  to  flip  the  polarity  of  the  lead_off  drive
based
      //on whether we're sensing the positive or negative inputs
      if (use_neg_inputs==false) {
           //we're using positive.  Set to default polarity
           ADS1299::WREG(LOFF_FLIP,0b00000000);delay(1);       //set    all
channels to zero
      } else {
           //we're using negative.  flip the polarity
           ADS1299::WREG(LOFF_FLIP,0b11111111);delay(1);
      }

    }

    //reset all the ADS1299's settings.  Call however you'd like.  Stops
all data acquisition
    void ADS1299Manager::reset(void)
    {
      ADS1299::RESET();               // send RESET command to default all
registers
```

```
      ADS1299::SDATAC();              // exit Read Data Continuous mode to
communicate with ADS

      delay(100);

      // turn off all channels
      for (int chan=1; chan <= OPENBCI_NCHAN_PER_BOARD; chan++) {
        deactivateChannel(chan);  //turn off the channel
        changeChannelLeadOffDetection(chan,OFF,BOTHCHAN);  //turn  off  any
impedance monitoring
      }

      setSRB1(use_SRB1());  //set whether SRB1 is active or not
      setAutoBiasGeneration(true);  //configure  ADS1299  so  that  bias  is
generated based on channel state
    };


    //deactivate the given channel...note: stops data colleciton to issue
its commands
    //  N is the channel number: 1-8
    //
    void ADS1299Manager::deactivateChannel(int N)
    {
      byte reg, config;

      //check the inputs
      if ((N < 1) || (N > OPENBCI_NCHAN_PER_BOARD)) return;

      //proceed...first, disable any data collection
      ADS1299::SDATAC(); delay(1);       // exit Read Data Continuous mode
to communicate with ADS

      //shut down the channel
      int    N_zeroRef    =    constrain(N-1,0,OPENBCI_NCHAN_PER_BOARD-1);
//subtracts 1 so that we're counting from 0, not 1
      reg = CH1SET+(byte)N_zeroRef;
      config = ADS1299::RREG(reg); delay(1);
      bitSet(config,7);   //left-most bit (bit 7) = 1, so this shuts down
the channel
      if  (use_neg_inputs)  bitClear(config,3);    //bit  3  =  0  disconnects
SRB2
      ADS1299::WREG(reg,config); delay(1);

      //set how this channel affects the bias generation...
      alterBiasBasedOnChannelState(N);
    };


    //Active a channel in single-ended mode
    //  N is 1 through 8
    //  gainCode is defined in the macros in the header file
    //  inputCode is defined in the macros in the header file
    void    ADS1299Manager::activateChannel(int    N,byte    gainCode,byte
inputCode)
    {
      byte reg, config;

       //check the inputs
      if ((N < 1) || (N > OPENBCI_NCHAN_PER_BOARD)) return;
```

```
      //proceed...first, disable any data collection
      ADS1299::SDATAC(); delay(1);       // exit Read Data Continuous mode
to communicate with ADS

      //active the channel using the given gain.   Set MUX for normal
operation
      //see ADS1299 datasheet, PDF p44
      N = constrain(N-1,0,OPENBCI_NCHAN_PER_BOARD-1);  //shift down by one
      byte configByte = 0b00000000;    //left-most zero (bit 7) is to
activate the channel
      gainCode = gainCode & 0b01110000;  //bitwise AND to get just the
bits we want and set the rest to zero
      configByte = configByte | gainCode; //bitwise OR to set just the
gain bits high or low and leave the rest alone
      inputCode = inputCode & 0b00000111;  //bitwise AND to get just the
bits we want and set the rest to zero
      configByte = configByte | inputCode; //bitwise OR to set just the
gain bits high or low and leave the rest alone
      if (use_SRB2[N]) configByte |= 0b00000000;    //set the SRB2
flag...p44 in the data sheet
      ADS1299::WREG(CH1SET+(byte)N,configByte); delay(1);

      //add this channel to the bias generation
      alterBiasBasedOnChannelState(N);

      // // Now, these actions are necessary whenever there is at least
one active channel
      // // though they don't strictly need to be done EVERY time we
activate a channel.
      // // just once after the reset.

      //activate SRB1 as the Negative input for all channels, if needed
      setSRB1(use_SRB1());

      //Finalize the bias setup...activate buffer and use internal
reference for center of bias creation, datasheet PDF p42
      //ADS1299::WREG(CONFIG3,0b11101100); delay(1);
        ADS1299::WREG(CONFIG3,0b11100000); delay(1);
    };

    //note that N here one-referenced (ie [1...N]), not [0...N-1]
    boolean ADS1299Manager::isChannelActive(int N_oneRef) {
         int N_zeroRef = constrain(N_oneRef-1,0,OPENBCI_NCHAN_PER_BOARD-
1);  //subtracts 1 so that we're counting from 0, not 1

         //get whether channel is active or not
         byte reg = CH1SET+(byte)N_zeroRef;
         byte config = ADS1299::RREG(reg); delay(1);
         boolean chanState = bitRead(config,7);
         return chanState;
    }

    void ADS1299Manager::setAutoBiasGeneration(boolean state) {
        use_channels_for_bias = state;

        //step through the channels are recompute the bias state
        for (int Ichan=1; Ichan<OPENBCI_NCHAN_PER_BOARD;Ichan++) {
            alterBiasBasedOnChannelState(Ichan);
        }
    }
```

```
     //note that N here one-referenced (ie [1...N]), not [0...N-1]
     void ADS1299Manager::alterBiasBasedOnChannelState(int N_oneRef) {
          int  N_zeroRef  =  constrain(N_oneRef-1,0,OPENBCI_NCHAN_PER_BOARD-
1);  //subtracts 1 so that we're counting from 0, not 1

          boolean activateBias = false;
          if ((use_channels_for_bias==true) && (isChannelActive(N_oneRef)))
{
               //activate this channel's bias
               activateBiasForChannel(N_oneRef);
          } else {
               deactivateBiasForChannel(N_oneRef);
          }
     }


     void ADS1299Manager::deactivateBiasForChannel(int N_oneRef) {
          int N_zeroRef = constrain(N_oneRef-1,0,OPENBCI_NCHAN_PER_BOARD-1);
//subtracts 1 so that we're counting from 0, not 1

          //deactivate this channel's bias...both positive and negative
          //see ADS1299 datasheet, PDF p44.
          byte reg, config;
          for (int I=0;I<2;I++) {
               if (I==0) {
                    reg = BIAS_SENSP;
               } else {
                    reg = BIAS_SENSN;
               }
               config = ADS1299::RREG(reg); delay(1);//get the current bias
settings
               bitClear(config,N_zeroRef);            //clear this channel's
bit to remove from bias generation
               ADS1299::WREG(reg,config);  delay(1);    //send  the  modified
byte back to the ADS
          }
     }
     void ADS1299Manager::activateBiasForChannel(int N_oneRef) {
          int N_zeroRef = constrain(N_oneRef-1,0,OPENBCI_NCHAN_PER_BOARD-1);
//subtracts 1 so that we're counting from 0, not 1

          //see ADS1299 datasheet, PDF p44.
          //per Chip's experiments, if using the P inputs, just include the
P inputs
          //per Joel's experiements, if using the N inputs, include both P
and N inputs
          byte reg, config;
          int nLoop = 1;  if (use_neg_inputs) nLoop=2;
          for (int i=0; i < nLoop; i++) {
               reg = BIAS_SENSP;
               if (i > 0) reg = BIAS_SENSN;
               config = ADS1299::RREG(reg); //get the current bias settings
               bitSet(config,N_zeroRef);                         //set  this
channel's bit
               ADS1299::WREG(reg,config);  delay(1);    //send  the  modified
byte back to the ADS
          }
     }
```

```
    //change the given channel's lead-off detection state...note: stops
data colleciton to issue its commands
    //  N is the channel number: 1-8
    //
    void    ADS1299Manager::changeChannelLeadOffDetection(int    N,    int
code_OFF_ON, int code_P_N_Both)
    {
      byte reg, config;

      //check the inputs
      if ((N < 1) || (N > OPENBCI_NCHAN_PER_BOARD)) return;
      N = constrain(N-1,0,OPENBCI_NCHAN_PER_BOARD-1);  //shift down by one

      //proceed...first, disable any data collection
      ADS1299::SDATAC(); delay(1);       // exit Read Data Continuous mode
to communicate with ADS

      if ((code_P_N_Both == PCHAN) || (code_P_N_Both == BOTHCHAN)) {
          //shut down the lead-off signal on the positive side
          reg = LOFF_SENSP;  //are we using the P inptus or the N inputs?
          config = ADS1299::RREG(reg); //get the current lead-off settings
          if (code_OFF_ON == OFF) {
              bitClear(config,N);                          //clear  this
channel's bit
          } else {
              bitSet(config,N);               //clear this channel's bit
          }
          ADS1299::WREG(reg,config); delay(1);   //send  the  modified  byte
back to the ADS
      }

      if ((code_P_N_Both == NCHAN) || (code_P_N_Both == BOTHCHAN)) {
          //shut down the lead-off signal on the negative side
          reg = LOFF_SENSN;  //are we using the P inptus or the N inputs?
          config = ADS1299::RREG(reg); //get the current lead-off settings
          if (code_OFF_ON == OFF) {
              bitClear(config,N);                          //clear  this
channel's bit
          } else {
              bitSet(config,N);               //clear this channel's bit
          }          //set this channel's bit
          ADS1299::WREG(reg,config); delay(1);   //send  the  modified  byte
back to the ADS
      }
    };

    void   ADS1299Manager::configureLeadOffDetection(byte   amplitudeCode,
byte freqCode)
    {
        amplitudeCode &= 0b00001100;  //only these two bits should be used
        freqCode &= 0b00000011;  //only these two bits should be used

        //get the current configuration of he byte
        byte reg, config;
        reg = LOFF;
        config = ADS1299::RREG(reg); //get the current bias settings

        //reconfigure the byte to get what we want
        config &= 0b11110000;  //clear out the last four bits
        config |= amplitudeCode;  //set the amplitude
        config |= freqCode;    //set the frequency
```

```
          //send the config byte back to the hardware
          ADS1299::WREG(reg,config);  delay(1);    //send  the  modified  byte
back to the ADS

     }

     void ADS1299Manager::setSRB1(boolean desired_state) {
          if (! desired_state) {
               ADS1299::WREG(MISC1,0b00100000);    delay(1);        //ADS1299
datasheet, PDF p46
          } else {
               ADS1299::WREG(MISC1,0b00000000);    delay(1);        //ADS1299
datasheet, PDF p46
          }
     }



     //Configure  the  test  signals  that  can  be  inernally  generated  by  the
ADS1299
     void  ADS1299Manager::configureInternalTestSignal(byte  amplitudeCode,
byte freqCode)
     {
          if   (amplitudeCode   ==   ADSTESTSIG_NOCHANGE)   amplitudeCode   =
(ADS1299::RREG(CONFIG2) & (0b00000100));
          if     (freqCode     ==     ADSTESTSIG_NOCHANGE)     freqCode     =
(ADS1299::RREG(CONFIG2) & (0b00000011));
          freqCode &= 0b00000011;  //only the last two bits should be used
          amplitudeCode &= 0b00000100;  //only this bit should be used
          byte message = 0b11000000 | freqCode | amplitudeCode;  //compose
the code

          ADS1299::WREG(CONFIG2,message); delay(1);

            //ADS1299::WREG(CONFIG2,0b11010000);delay(1);     //set  internal
test signal, default amplitude, default speed, datasheet PDF Page 41
           //ADS1299::WREG(CONFIG2,0b11010001);delay(1);     //set  internal
test signal, default amplitude, 2x speed, datasheet PDF Page 41
           //ADS1299::WREG(CONFIG2,0b11010101);delay(1);     //set  internal
test signal, 2x amplitude, 2x speed, datasheet PDF Page 41
           //ADS1299::WREG(CONFIG2,0b11010011);  delay(1);    //set  internal
test signal, default amplitude, at DC, datasheet PDF Page 41
           //ADS1299::WREG(CONFIG3,0b01101100);  delay(1);    //use  internal
reference for center of bias creation, datasheet PDF p42
     }

     //Start continuous data acquisition
     void ADS1299Manager::start(void)
     {
          ADS1299::RDATAC();  delay(1);                  //  enter  Read  Data
Continuous mode
          ADS1299::START();    //start the data acquisition
     }

     //Query to see if data is available from the ADS1299...return TRUE is
data is available
     int ADS1299Manager::isDataAvailable(void)
     {
       return (!(digitalRead(PIN_DRDY)));
```

```
    }

    //Stop the continuous data acquisition
    void ADS1299Manager::stop(void)
    {
        ADS1299::STOP(); delay(1);   //start the data acquisition
        ADS1299::SDATAC(); delay(1);        // exit Read Data Continuous
mode to communicate with ADS
    }

    //print as text each channel's data
    //    print channels 1-N (where N is 1-8...anything else will return
with no action)
    //    sampleNumber is a number that, if greater than zero, will be
printed at the start of the line
    void    ADS1299Manager::printChannelDataAsText(int    N,    long    int
sampleNumber)
    {
        //check the inputs
        if ((N < 1) || (N > n_chan_all_boards)) return;

        //print the sample number, if not disabled
        if (sampleNumber > 0) {
            Serial.print(sampleNumber);
            Serial.print(", ");
        }

        //print each channel
        for (int chan = 0; chan < N; chan++ )
        {
            Serial.print(channelData[chan]);
            Serial.print(", ");
        }

        //print end of line
        Serial.println();
    };


    //write as binary each channel's data
    //    print channels 1-N (where N is 1-8...anything else will return
with no action)
    //    sampleNumber is a number that, if greater than zero, will be
printed at the start of the line
    int32 val;
    byte *val_ptr = (byte *)(&val);
    void     ADS1299Manager::writeChannelDataAsBinary(int     N,     long
sampleNumber){
        ADS1299Manager::writeChannelDataAsBinary(N,sampleNumber,false,0,fa
lse);
    }
    void     ADS1299Manager::writeChannelDataAsBinary(int     N,     long
sampleNumber,boolean useSyntheticData){
        ADS1299Manager::writeChannelDataAsBinary(N,sampleNumber,false,0,us
eSyntheticData);
    }
    void     ADS1299Manager::writeChannelDataAsBinary(int     N,     long
sampleNumber,long int auxValue){
        ADS1299Manager::writeChannelDataAsBinary(N,sampleNumber,true,auxVa
lue,false);
    }
```

```
    void      ADS1299Manager::writeChannelDataAsBinary(int      N,      long
sampleNumber,long int auxValue, boolean useSyntheticData){
         ADS1299Manager::writeChannelDataAsBinary(N,sampleNumber,true,auxVa
lue,useSyntheticData);
    }
    void      ADS1299Manager::writeChannelDataAsBinary(int      N,      long
sampleNumber,boolean sendAuxValue,
         long int auxValue, boolean useSyntheticData)
    {
         //check the inputs
         if ((N < 1) || (N > n_chan_all_boards)) return;

         // Write start byte
         Serial.write( (byte) PCKT_START);

         //write the length of the payload
         //byte byte_val = (1+8)*4;
         byte payloadBytes = (byte)((1+N)*4);     //length of data payload,
bytes
         if (sendAuxValue) payloadBytes+= (byte)4;   //add four more bytes
for the aux value
         Serial.write(payloadBytes);  //write the payload length

         //write the sample number, if not disabled
         val = sampleNumber;
         Serial.write(val_ptr,4); //4 bytes long

         //write each channel
         for (int chan = 0; chan < N; chan++ )
         {
              //get this channel's data
              if (useSyntheticData) {
                   val = makeSyntheticSample(sampleNumber,chan);
                   //val = sampleNumber;
              } else {
                   //get the real EEG data for this channel
                   val = channelData[chan];
              }
              Serial.write(val_ptr,4); //4 bytes long
         }

         // Write the AUX value
         if (sendAuxValue) {
              val = auxValue;
              Serial.write(val_ptr,4); //4 bytes long
         }

         // Write footer
         Serial.write((byte)PCKT_END);

         // force everything out
         //Serial.flush();
    };

    //write channel data using binary format of ModularEEG so that it can
be used by BrainBay (P2 protocol)
    //this only sends 6 channels of data, per the P2 protocol
    //http://www.shifz.org/brainbay/manuals/brainbay_developer_manual.pdf
    #define max_int16 (32767)
    #define min_int16 (-32767)
    void ADS1299Manager::writeChannelDataAsOpenEEG_P2(long sampleNumber) {
```

```
        ADS1299Manager::writeChannelDataAsOpenEEG_P2(sampleNumber,false);
    }
    void                ADS1299Manager::writeChannelDataAsOpenEEG_P2(long
sampleNumber,boolean useSyntheticData) {
        static int count = -1;
        byte sync0 = 0xA5;
        byte sync1 = 0x5A;
        byte version = 2;

        Serial.write(sync0);
        Serial.write(sync1);
        Serial.write(version);
        byte foo = (byte)sampleNumber;
        if (foo == sync0) foo--;
        Serial.write(foo);

        long val32; //32-bit
        int val_i16;  //16-bit
        unsigned int val_u16;  //16-bit
        byte *val16_ptr = (byte *)(&val_u16);  //points to the memory for
the variable above
        for (int chan = 0; chan < 6; chan++ )
        {
            //get this channel's data
            if (useSyntheticData) {
                //generate XX uV pk-pk signal
                //long    time_samp_255    =    (long)((sampleNumber)    &
(0x000000FF));  //make an 8-bit ramp waveform
                //time_samp_255  =  (long)((time_samp_255*(long)(chan+1))
& (0x000000FF)); //each channel is faster than the previous
                //time_samp_255    +=    256L*2L;         //make    zero
mean...empirically tuned via BrainBay visualization
                //val32 = (synthetic_amplitude_counts * time_samp_255) /
255L; //scaled zero-mean ramp
                val32 = makeSyntheticSample(sampleNumber,chan) + 127L +
256L*2L;  //make zero mean...empirically tuned via BrainBay visualization
            } else {
                //get the real EEG data for this channel
                val32 = channelData[chan];
            }

            //prepare the value for transmission
            val32 = val32 / (32);  //shrink to fit within a 16-bit number
            val32 = constrain(val32,min_int16,max_int16);  //constrain to
fit in 16 bits
            val_u16 = (unsigned int) (val32 & (0x0000FFFF));  //truncate
and cast
            if (val_u16 > 1023) val_u16 = 1023;

            //Serial.write(val16_ptr,2);  //low  byte  than  high  byte  on
Arduino
            //Serial.write((byte)((val_u16 >> 8) & 0x00FF)); //high byte
            //Serial.write((byte)(val_u16 & 0x00FF)); //low byte
            foo = (byte)((val_u16 >> 8) & 0x00FF); //high byte
            if (foo == sync0) foo--;
            Serial.write(foo);
            foo = (byte)(val_u16 & 0x00FF); //high byte
            if (foo == sync0) foo--;
            Serial.write(foo);
```

```
        }
        //byte switches = 0b00000000;  //the last thing required by the P2
data protocol
        byte switches = 0x07;
        count++; if (count >= 18) count=0;
        if (count >= 9) {
             switches = 0x0F;
        }
        Serial.write(switches);
    }

    #define  synthetic_amplitude_counts  (8950L)       //counts  peak-to-
peak...should be 200 uV pk-pk  2.0*(100e-6 / (4.5 / 24 / 2^24))
    long  int  ADS1299Manager::makeSyntheticSample(long  sampleNumber,int
chan) {
        //generate XX uV pk-pk signal
        long  time_samp_255  =  (long)((sampleNumber)  &  (0x000000FF));
//make an 8-bit ramp waveform
        time_samp_255    =    (long)((time_samp_255*(long)(chan+1))    &
(0x000000FF)); //each channel is faster than the previous
        //time_samp_255 += 256L*2L;  //make zero mean...empirically tuned
via BrainBay visualization
        time_samp_255 -= 127L;
        return  (synthetic_amplitude_counts  *  time_samp_255)  /  255L;
//scaled zero-mean ramp
    };

    //print out the state of all the control registers
    void ADS1299Manager::printAllRegisters(void)
    {
        boolean prevVerboseState = verbose;

           verbose = true;
           ADS1299::RREGS(0x00,0x10);     // write the first registers
           delay(100);  //stall to let all that data get read by the PC
           ADS1299::RREGS(0x11,0x17-0x11);     // write the rest
           verbose = prevVerboseState;
    }

    //only use SRB1 if all use_SRB2 are set to false
    boolean ADS1299Manager::use_SRB1(void) {
        for (int Ichan=0; Ichan < OPENBCI_NCHAN_PER_BOARD; Ichan++) {
            if (use_SRB2[Ichan]) {
                return false;
            }
        }
        return true;
    }
```

Pasta ADS1299: Ficheiro “ADS1299Manager.h”

```
/*-----------------------------------------------------------------------
-----------------------------------------------------------------
* File Name: ADS1299Manager.h
* Author: Chip Audette
* Created in: 2013
```

```
* Modified: 2014 by Miguel Sousa
------------------------------------------------------------------------------
------------------------------------------------------------------*/

#ifndef ____ADS1299Manager__
#define ____ADS1299Manager__

#include <ADS1299.h>

//Pick which version of OpenBCI you have
#define OPENBCI_V1 (1)    //Sept 2013
#define OPENBCI_V2 (2)    //Oct 24, 2013
#define OPENBCI_NCHAN_PER_BOARD (8)  // number of EEG channels

/*   Arduino Uno - Pin Assignments
  SCK = 13
  MISO [DOUT] = 12
  MOSI [DIN] = 11
  CS = 10;
  RESET = 9;
  DRDY = 8;
*/
#define PIN_DRDY (8)
#define PIN_RST (9)
#define PIN_CS (10)
#define SCK_MHZ (4)

//gainCode choices
#define ADS_GAIN01 (0b00000000)
#define ADS_GAIN02 (0b00010000)
#define ADS_GAIN04 (0b00100000)
#define ADS_GAIN06 (0b00110000)
#define ADS_GAIN08 (0b01000000)
#define ADS_GAIN12 (0b01010000)
#define ADS_GAIN24 (0b01100000)

//inputCode choices
#define ADSINPUT_NORMAL (0b00000000)
#define ADSINPUT_SHORTED (0b00000001)
#define ADSINPUT_TESTSIG (0b00000101)

//test signal choices...ADS1299 datasheet page 41
#define ADSTESTSIG_AMP_1X (0b00000000)
#define ADSTESTSIG_AMP_2X (0b00000100)
#define ADSTESTSIG_PULSE_SLOW (0b00000000)
#define ADSTESTSIG_PULSE_FAST (0b00000001)
#define ADSTESTSIG_DCSIG (0b00000011)
#define ADSTESTSIG_NOCHANGE (0b11111111)

//Lead-off signal choices
#define LOFF_MAG_6NA (0b00000000)
#define LOFF_MAG_24NA (0b00000100)
#define LOFF_MAG_6UA (0b00001000)
#define LOFF_MAG_24UA (0b00001100)
#define LOFF_FREQ_DC (0b00000000)
#define LOFF_FREQ_7p8HZ (0b00000001)
#define LOFF_FREQ_31p2HZ (0b00000010)
#define LOFF_FREQ_FS_4 (0b00000011)
#define PCHAN (1)
#define NCHAN (2)
#define BOTHCHAN (3)
```

```
#define OFF (0)
#define ON (1)

//binary communication codes for each packet
#define PCKT_START 0xA0
#define PCKT_END 0xC0

class ADS1299Manager : public ADS1299 {
  public:
    void initialize(void);                                          //initialize
the ADS1299 controller.  Call once.  Assumes OpenBCI_V2
    void initialize(int version,boolean isDaisy);                   //initialize
the ADS1299 controller.  Call once.  Set which version of OpenBCI you're
using.
    void setVersionOpenBCI(int version);            //Set which version of
OpenBCI you're using.
    void reset(void);                                               //reset all
the ADS1299's settings.  Call however you'd like
    boolean isChannelActive(int N_oneRef);
    void  activateChannel(int  N_oneRef,  byte  gainCode,byte  inputCode);
//setup the channel 1-8
    void                  deactivateChannel(int                  N_oneRef);
//disable given channel 1-8
    void  configureLeadOffDetection(byte  amplitudeCode,  byte  freqCode);
//configure the lead-off detection signal parameters
    void changeChannelLeadOffDetection(int N_oneRef, int code_OFF_ON, int
code_P_N_Both);
    void  configureInternalTestSignal(byte  amplitudeCode,  byte  freqCode);
//configure the test signal parameters
    void start(void);
    void stop(void);
    int isDataAvailable(void);
    void printChannelDataAsText(int N, long int sampleNumber);
    void writeChannelDataAsBinary(int N, long int sampleNumber);
    void  writeChannelDataAsBinary(int  N,  long  int  sampleNumber,  boolean
useSyntheticData);
    void  writeChannelDataAsBinary(int  N,  long  int  sampleNumber,  long  int
auxValue);
    void  writeChannelDataAsBinary(int  N,  long  int  sampleNumber,  long  int
auxValue, boolean useSyntheticData);
    void  writeChannelDataAsBinary(int  N,  long  int  sampleNumber,  boolean
sendAuxValue,long int auxValue, boolean useSyntheticData);
    void writeChannelDataAsOpenEEG_P2(long int sampleNumber);
    void   writeChannelDataAsOpenEEG_P2(long   int   sampleNumber,   boolean
useSyntheticData);
    void printAllRegisters(void);
    void setSRB1(boolean desired_state);
    void alterBiasBasedOnChannelState(int N_oneRef);
    void deactivateBiasForChannel(int N_oneRef);
    void activateBiasForChannel(int N_oneRef);
    void setAutoBiasGeneration(boolean state);


  private:
    boolean use_neg_inputs;
    boolean use_SRB2[OPENBCI_NCHAN_PER_BOARD];
    boolean use_channels_for_bias;
    boolean use_SRB1(void);
    long int makeSyntheticSample(long sampleNumber,int chan);
    int n_chan_all_boards;
```

```
};

#endif
```

Pasta ADS1299: Ficheiro “ADS1299.cpp”

```
/*-----------------------------------------------------------------------
-----------------------------------------------------------------
* File Name: ADS1299.cpp
* Author: Conor Russomanno, Luke Travis, and Joel Murphy and
* Created in: 2013
* Modified: 2014 by Chip Audette and Miguel Sousa
-----------------------------------------------------------------------
-----------------------------------------------------------------*/

#include "pins_arduino.h"
#include "ADS1299.h"

void ADS1299::initialize(int _DRDY, int _RST, int _CS, int _FREQ, boolean
_isDaisy){
     isDaisy = _isDaisy;
     DRDY = _DRDY;
     CS = _CS;
     int FREQ = _FREQ;
     int RST = _RST;

     delay(50);                  // recommended power up sequence requiers
Tpor (~32mS)
     pinMode(RST,OUTPUT);
     pinMode(RST,LOW);
     delayMicroseconds(4);   // toggle reset pin
     pinMode(RST,HIGH);
     delayMicroseconds(20); // recommended to wait 18 Tclk before using
device (~8uS);


    // **** ----- SPI Setup ----- **** //

    // Set direction register for SCK and MOSI pin.
    // MISO pin automatically overrides to INPUT.
    // When the SS pin is set as OUTPUT, it can be used as
    // a general purpose output port (it doesn't influence
    // SPI operations).

    pinMode(SCK, OUTPUT);
    pinMode(MOSI, OUTPUT);
    pinMode(SS, OUTPUT);


    digitalWrite(SCK, LOW);
    digitalWrite(MOSI, LOW);
    digitalWrite(SS, HIGH);

    // set as master and enable SPI
    SPCR |= _BV(MSTR);
    SPCR |= _BV(SPE);
    //set bit order
    SPCR &= ~(_BV(DORD)); ////SPI data format is MSB (pg. 25)
     // set data mode
```

```
    SPCR = (SPCR & ~SPI_MODE_MASK) | SPI_DATA_MODE; //clock polarity = 0;
clock phase = 1 (pg. 8)
    // set clock divider
     switch (FREQ){
          case 8:
               DIVIDER = SPI_CLOCK_DIV_2;
               break;
          case 4:
               DIVIDER = SPI_CLOCK_DIV_4;
               break;
          case 1:
               DIVIDER = SPI_CLOCK_DIV_16;
               break;
          default:
               break;
     }
    SPCR = (SPCR & ~SPI_CLOCK_MASK) | (DIVIDER);  // set SCK frequency
    SPSR = (SPSR & ~SPI_2XCLOCK_MASK) | (DIVIDER); // by dividing 16MHz
system clock

// **** ----- End of SPI Setup ----- **** //

    // initalize the  data ready chip select and reset pins:
    pinMode(DRDY, INPUT);
    pinMode(CS, OUTPUT);

     digitalWrite(CS,HIGH);
     digitalWrite(RST,HIGH);
}

//System Commands
void ADS1299::WAKEUP() {
    digitalWrite(CS, LOW);
    transfer(_WAKEUP);
    digitalWrite(CS, HIGH);
    delayMicroseconds(3);        //must wait 4 tCLK cycles before sending
another command (Datasheet, pg. 35)
}

void ADS1299::STANDBY() {        //  only allowed to send WAKEUP after
sending STANDBY
    digitalWrite(CS, LOW);
    transfer(_STANDBY);
    digitalWrite(CS, HIGH);
}

void ADS1299::RESET() {          //  reset all the registers to default
settings
    digitalWrite(CS, LOW);
    transfer(_RESET);
    delayMicroseconds(12);       //must wait 18 tCLK cycles to execute this
command (Datasheet, pg. 35)
    digitalWrite(CS, HIGH);
}

void ADS1299::START() {          //start data conversion
    digitalWrite(CS, LOW);
    transfer(_START);
    digitalWrite(CS, HIGH);
}
```

```
void ADS1299::STOP() {           //stop data conversion
    digitalWrite(CS, LOW);
    transfer(_STOP);
    digitalWrite(CS, HIGH);
}

void ADS1299::RDATAC() {
    digitalWrite(CS, LOW);
    transfer(_RDATAC);
    digitalWrite(CS, HIGH);
     delayMicroseconds(3);
}
void ADS1299::SDATAC() {
    digitalWrite(CS, LOW);
    transfer(_SDATAC);
    digitalWrite(CS, HIGH);
     delayMicroseconds(3);   //must wait 4 tCLK cycles after executing this
command (Datasheet, pg. 37)
}


// Register Read/Write Commands
byte ADS1299::getDeviceID() {              // simple hello world com check
     byte data = RREG(0x00);
     if(verbose){                          // verbose otuput
          Serial.print(F("Device ID "));
          printHex(data);
     }
     return data;
}

byte ADS1299::RREG(byte _address) {        //     reads   ONE   register   at
_address
    byte opcode1 = _address + 0x20;   //  RREG expects 001rrrrr where rrrrr
= _address
    digitalWrite(CS, LOW);                 //  open SPI
    transfer(opcode1);                     //  opcode1
    transfer(0x00);                        //  opcode2
    regData[_address]  =  transfer(0x00);//    update  mirror  location  with
returned byte
    digitalWrite(CS, HIGH);           //  close SPI
     if (verbose){                         //  verbose output
          printRegisterName(_address);
          printHex(_address);
          Serial.print(", ");
          printHex(regData[_address]);
          Serial.print(", ");
          for(byte j = 0; j<8; j++){
               Serial.print(bitRead(regData[_address], 7-j));
               if(j!=7) Serial.print(", ");
          }

          Serial.println();
     }
     return regData[_address];             //   return   requested   register
value
}

// Read more than one register starting at _address
void ADS1299::RREGS(byte _address, byte _numRegistersMinusOne) {
//   for(byte i = 0; i < 0x17; i++){
```

```
//        regData[i] = 0;                        //  reset the regData array
//    }
    byte opcode1 = _address + 0x20;   //  RREG expects 001rrrrr where rrrrr
= _address
    digitalWrite(CS, LOW);                 //  open SPI
    transfer(opcode1);                     //  opcode1
    transfer(_numRegistersMinusOne);  //  opcode2
    for(int i = 0; i <= _numRegistersMinusOne; i++){
        regData[_address + i] = transfer(0x00);     //    add  register  byte
to mirror array
        }
    digitalWrite(CS, HIGH);           //  close SPI
     if(verbose){                          //  verbose output
         for(int i = 0; i<= _numRegistersMinusOne; i++){
              printRegisterName(_address + i);
              printHex(_address + i);
              Serial.print(", ");
              printHex(regData[_address + i]);
              Serial.print(", ");
              for(int j = 0; j<8; j++){
                   Serial.print(bitRead(regData[_address + i], 7-j));
                   if(j!=7) Serial.print(", ");
              }
              Serial.println();
         }
    }

}

void ADS1299::WREG(byte _address, byte _value) {    //   Write  ONE  register
at _address
    byte opcode1 = _address + 0x40;   //  WREG expects 010rrrrr where rrrrr
= _address
    digitalWrite(CS, LOW);                 //  open SPI
    transfer(opcode1);                     //  Send WREG command & address
    transfer(0x00);                         //   Send      number      of
registers to read -1
    transfer(_value);                      //    Write   the   value   to   the
register
    digitalWrite(CS, HIGH);           //  close SPI
     regData[_address] = _value;           //  update the mirror array
     if(verbose){                          //  verbose output
         Serial.print(F("Register "));
         printHex(_address);
         Serial.println(F(" modified."));
     }
}

void ADS1299::WREGS(byte _address, byte _numRegistersMinusOne) {
    byte opcode1 = _address + 0x40;        //   WREG  expects  010rrrrr  where
rrrrr = _address
    digitalWrite(CS, LOW);                 //  open SPI
    transfer(opcode1);                     //  Send WREG command & address
    transfer(_numRegistersMinusOne);  //   Send number of registers to read
-1
     for (int i=_address; i <=(_address + _numRegistersMinusOne); i++){
         transfer(regData[i]);             //  Write to the registers
     }
     digitalWrite(CS,HIGH);                //  close SPI
     if(verbose){
         Serial.print(F("Registers "));
```

```
            printHex(_address); Serial.print(F(" to "));
            printHex(_address + _numRegistersMinusOne);
            Serial.println(F(" modified"));
        }
}


void ADS1299::updateChannelData(){
     byte inByte;
     int nchan=8;  //assume 8 channel.  If needed, it automatically changes
to 16 automatically in a later block.
     digitalWrite(CS, LOW);                //  open SPI

     // READ CHANNEL DATA FROM FIRST ADS IN DAISY LINE
     for(int i=0; i<3; i++){           //  read 3 byte status register from
ADS 1 (1100+LOFF_STATP+LOFF_STATN+GPIO[7:4])
          inByte = transfer(0x00);
          stat_1 = (stat_1<<8) | inByte;
     }

     for(int i = 0; i<8; i++){
          for(int j=0; j<3; j++){      //  read 24 bits of channel data from
1st ADS in 8 3 byte chunks
               inByte = transfer(0x00);
               channelData[i] = (channelData[i]<<8) | inByte;
          }
     }

     if (isDaisy) {
          nchan = 16;
          // READ CHANNEL DATA FROM SECOND ADS IN DAISY LINE
          for(int i=0; i<3; i++){           //  Read 3 byte status register
from ADS 2 (1100+LOFF_STATP+LOFF_STATN+GPIO[7:4])
               inByte = transfer(0x00);
               stat_2 = (stat_1<<8) | inByte;
          }

          for(int i = 8; i<16; i++){
               for(int j=0; j<3; j++){      // Read 24 bits of channel
                                            data from 2nd ADS in 8 3
                                            byte chunks
                    inByte = transfer(0x00);
                    channelData[i] = (channelData[i]<<8) | inByte;
               }
          }
     }

     digitalWrite(CS, HIGH);               //  close SPI

//-------------------------Reformat the Numbers-------------------------

     for(int i=0; i<nchan; i++){       // Convert 3 byte 2's compliment
                                          to 4 byte 2's compliment
          if(bitRead(channelData[i],23) == 1){
               channelData[i] |= 0xFF000000;
          }else{
               channelData[i] &= 0x00FFFFFF;
          }
     }
}
```

```
//------------------------------Read Data-------------------------------

void ADS1299::RDATA() {      //  Use in Stop Read Continuous  mode when
                                DRDY goes low
     byte inByte;
     stat_1 = 0;             //  Clear the status registers
     stat_2 = 0;
     int nchan = 8;          // Assume 8 channel.
                            If needed, it automatically changes to 16
                            automatically in a later block.
     digitalWrite(CS, LOW); // Open SPI
     transfer(_RDATA);

//-----------READ CHANNEL DATA FROM FIRST ADS IN DAISY LINE-----------

     for(int i=0; i<3; i++){      // Read 3 byte status register:
                                     (1100+LOFF_STATP+LOFF_STATN+GPIO[7:4])
          inByte = transfer(0x00);
          stat_1 = (stat_1<<8) | inByte;
     }

     for(int i = 0; i<8; i++){
          for(int j=0; j<3; j++){ // Read 24 bits of channel data  from
                                                1st ADS in 8 3 byte chunks
               inByte = transfer(0x00);
               channelData[i] = (channelData[i]<<8) | inByte;
          }
     }

     if (isDaisy) {
          nchan = 16;

//-----------READ CHANNEL DATA FROM SECOND ADS IN DAISY LINE----------

    for(int i=0; i<3; i++){                  // Read 3 byte status register
                                        (1100+LOFF_STATP+LOFF_STATN+GPIO[
                                                                7:4])
               inByte = transfer(0x00);
               stat_2 = (stat_1<<8) | inByte;
          }

          for(int i = 8; i<16; i++){
                    for(int j=0; j<3; j++){      //  Read 24 bits of channel
                                                    data from 2nd ADS in 8 3
                                                              byte chunks
                    inByte = transfer(0x00);
                    channelData[i] = (channelData[i]<<8) | inByte;
               }
          }
     }

     for(int i=0; i<nchan; i++){      // convert 3 byte 2's compliment
     to 4 byte 2's compliment
          if(bitRead(channelData[i],23) == 1){
               channelData[i] |= 0xFF000000;
          }else{
               channelData[i] &= 0x00FFFFFF;
          }
     }
}
```

```
//--------------String-Byte converters for RREG and WREG--------------

void ADS1299::printRegisterName(byte _address) {
    if(_address == ID){
        Serial.print(F("ID, ")); //the "F" macro loads the string directly
                                 from Flash memory, thereby saving RAM
    }
    else if(_address == CONFIG1){
        Serial.print(F("CONFIG1, "));
    }
    else if(_address == CONFIG2){
        Serial.print(F("CONFIG2, "));
    }
    else if(_address == CONFIG3){
        Serial.print(F("CONFIG3, "));
    }
    else if(_address == LOFF){
        Serial.print(F("LOFF, "));
    }
    else if(_address == CH1SET){
        Serial.print(F("CH1SET, "));
    }
    else if(_address == CH2SET){
        Serial.print(F("CH2SET, "));
    }
    else if(_address == CH3SET){
        Serial.print(F("CH3SET, "));
    }
    else if(_address == CH4SET){
        Serial.print(F("CH4SET, "));
    }
    else if(_address == CH5SET){
        Serial.print(F("CH5SET, "));
    }
    else if(_address == CH6SET){
        Serial.print(F("CH6SET, "));
    }
    else if(_address == CH7SET){
        Serial.print(F("CH7SET, "));
    }
    else if(_address == CH8SET){
        Serial.print(F("CH8SET, "));
    }
    else if(_address == BIAS_SENSP){
        Serial.print(F("BIAS_SENSP, "));
    }
    else if(_address == BIAS_SENSN){
        Serial.print(F("BIAS_SENSN, "));
    }
    else if(_address == LOFF_SENSP){
        Serial.print(F("LOFF_SENSP, "));
    }
    else if(_address == LOFF_SENSN){
        Serial.print(F("LOFF_SENSN, "));
    }
    else if(_address == LOFF_FLIP){
        Serial.print(F("LOFF_FLIP, "));
    }
    else if(_address == LOFF_STATP){
        Serial.print(F("LOFF_STATP, "));
    }
```

```
    else if(_address == LOFF_STATN){
        Serial.print(F("LOFF_STATN, "));
    }
    else if(_address == GPIO){
        Serial.print(F("GPIO, "));
    }
    else if(_address == MISC1){
        Serial.print(F("MISC1, "));
    }
    else if(_address == MISC2){
        Serial.print(F("MISC2, "));
    }
    else if(_address == CONFIG4){
        Serial.print(F("CONFIG4, "));
    }
}

//----------------------SPI communication methods--------------------

byte ADS1299::transfer(byte _data) {
     cli();
    SPDR = _data;
    while (!(SPSR & _BV(SPIF)))
        ;
     sei();
    return SPDR;
}

//-----------Used for printing HEX in verbose feedback mode-----------

void ADS1299::printHex(byte _data){
     Serial.print("0x");
    if(_data < 0x10) Serial.print("0");
    Serial.print(_data, HEX);
}
```

Pasta ADS1299: Ficheiro “ADS1299.h”

```
/*-------------------------------------------------------------------------
-------------------------------------------------------------

* File Name: ADS1299.h
* Author: Conor Russomanno, Luke Travis, and Joel Murphy and
* Created in: 2013
* Modified: 2014 by Chip Audette and Miguel Sousa
* Description: Part of the Arduino Library

-------------------------------------------------------------------------
-------------------------------------------------------------*/

#ifndef ____ADS1299__
#define ____ADS1299__

#include <stdio.h>
#include <Arduino.h>
#include <avr/pgmspace.h>
#include "Definitions.h"

class ADS1299 {
```

```
public:
    void initialize(int _DRDY, int _RST, int _CS, int _FREQ, boolean
_isDaisy);

//----------ADS1299 SPI Command Definitions (Datasheet, p35)----------
//-------------------------System Commands-------------------------

    void WAKEUP();
    void STANDBY();
    void RESET();
    void START();
    void STOP();

//-----------------------Data Read Commands-----------------------

    void RDATAC();
    void SDATAC();
    void RDATA();

//--------------------Register Read/Write Commands--------------------

    byte getDeviceID();
    byte RREG(byte _address);
    void RREGS(byte _address, byte _numRegistersMinusOne);
    void printRegisterName(byte _address);
    void WREG(byte _address, byte _value);
    void WREGS(byte _address, byte _numRegistersMinusOne);
    void printHex(byte _data);
    void updateChannelData();

//------------------------SPI Transfer function----------------------

     byte transfer(byte _data);

//---------------------------Configuration---------------------------

    int DRDY, CS;           // pin numbers for DRDY and CS
    int DIVIDER;            // select SPI SCK frequency
    int stat_1, stat_2;     //  used to hold the status register for
                            boards 1 and 2
    byte regData [24];      // array is used to mirror register data
    long channelData [16];  //  array used when reading channel data
                            board 1+2
    boolean verbose;        // turn on/off Serial feedback
    boolean isDaisy;        // does this have a daisy chain board?

};
#endif
```

Pasta ADS1299: Ficheiro “Definitions.h”

```
/*-------------------------------------------------------------------------
-----------------------------------------------------------------

* File Name: Definitions.h
* Author: Conor Russomanno, Luke Travis, and Joel Murphy and
* Created in: 2013
* Modified: 2014 by and Miguel Sousa
* Description: SPI Command Definition and Register Addresses
```

```
-------------------------------------------------------------------------
-------------------------------------------------------------*/

#ifndef _Definitions_h
#define _Definitions_h
#define SPI_DATA_MODE 0x04       // CPOL = 0; CPHA = 1 (Datasheet, p8)
#define SPI_MODE_MASK 0x0C       // Mask of CPOL and CPHA  on SPCR
#define SPI_CLOCK_MASK 0x03      // SPR1 = bit 1, SPR0 = bit 0 on SPCR
#define SPI_2XCLOCK_MASK 0x01    // SPI2X = bit 0 on SPSR
#define SPI_CLOCK_DIV_2 0X04     // 8MHz SPI SCK
#define SPI_CLOCK_DIV_4 0X00     // 4MHz SPI SCK
#define SPI_CLOCK_DIV_16 0x01    // 1MHz SPI SCK

//------SPI Command Definition Byte Assignments (Datasheet, p35)------

#define _WAKEUP 0x02    // Wake-up from standby mode
#define _STANDBY 0x04   // Enter Standby mode
#define _RESET 0x06     // Reset the device registers to default
#define _START 0x08     // Start and restart (synchronize) conversions
#define _STOP 0x0A      // Stop conversion
#define _RDATAC 0x10    // Enable Read Data Continuous mode (default mode
                        at power-up)
#define _SDATAC 0x11    // Stop Read Data Continuous mode
#define _RDATA 0x12     // Read data by command; supports multiple read
                        back

//-------------------------Register Addresses-------------------------

#define ID 0x00
#define CONFIG1 0x01
#define CONFIG2 0x02
#define CONFIG3 0x03
#define LOFF 0x04
#define CH1SET 0x05
#define CH2SET 0x06
#define CH3SET 0x07
#define CH4SET 0x08
#define CH5SET 0x09
#define CH6SET 0x0A
#define CH7SET 0x0B
#define CH8SET 0x0C
#define BIAS_SENSP 0x0D
#define BIAS_SENSN 0x0E
#define LOFF_SENSP 0x0F
#define LOFF_SENSN 0x10
#define LOFF_FLIP 0x11
#define LOFF_STATP 0x12
#define LOFF_STATN 0x13
#define GPIO 0x14
#define MISC1 0x15
#define MISC2 0x16
#define CONFIG4 0x17
#endif
```

Anexo J – Dados Demográficos dos Utilizadores do Protótipo II e dos dispositivos *Enobio NE* e *g.MOBILab+*

Na avaliação do desempenho do Protótipo II, nomeadamente na obtenção de sinais de EEG utilizando o Método *Motor Imagery*, foram recolhidos alguns dados demográficos de cada utilizador, os quais podem ser observados na Tabela J. 1.

Tabela J. 1– Dados demográficos de cada utilizador do Protótipo II.

Utilizador	Sexo	Idade	Nível de Estudos	Destreza
1	Masculino	25-34	Mestrado	Mão Direita
2	Masculino	25-34	Licenciatura	Mão Direita
3	Masculino	18-24	Licenciatura	Mão Direita
4	Masculino	25-34	Licenciatura	Mão Direita
5	Masculino	35-44	Mestrado	Mão Direita
6	Masculino	25-34	Licenciatura	Mão Direita
7	Masculino	25-34	Licenciatura	Mão Direita
8	Masculino	25-34	Licenciatura	Mão Direita