UNIVERSIDADE FEDERAL DO PARANÁ
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

AQUISIÇÃO E PRÉ-PROCESSAMENTO DE ONDAS EEG.

CURITIBA
2019

MARCOS HENRIQUE HORST CAVALLI GRR20143474

AQUISIÇÃO E PRÉ-PROCESSAMENTO DE ONDAS EEG.

Monografia apresentada como trabalho de conclusão de curso B do curso de graduação de Engenharia Elétrica, Setor de Tecnologia, Departamento de Engenharia Elétrica, Universidade Federal do Paraná.

Orientador: Prof. Dr. José Carlos da Cunha

CURITIBA

2019

Este trabalho é dedicado às crianças adultas que
sempre buscam meios para realizar suas ideias,
nunca se limitando pela impossibilidade de realizá-las.

AGRADECIMENTOS

Há alguns anos, enquanto estava pesquisando sobre tecnologias diferentes que envolviam sinais eletroencefalográficos, inspirado por um programa de televisão que havia visto na época, decidi que gostaria de mexer com isto. Na mesma semana, procurei o professor Dr. José Carlos da Cunha, que me alertou que não seria uma área fácil, mas que seria possível desenvolver algo na área.

Os anos passaram mas a vontade de desenvolver este projeto continuou. Quando chegou a época de TCC A, entrei em contato com o professor e ele aceitou orientar eu e um amigo a realizar um projeto na área. Meus primeiros agradecimentos são ao professor Dr. José Carlos da Cunha, pela ótima orientação e auxilio durante o período do desenvolvimento desta monografia e ao amigo Rodolfo Tureck, que me apoiou no início deste projeto e me ajudou a alavanca-lo.

Seguindo com o decorrer do projeto, inúmeras dificuldades foram surgindo, como imaginado no início com isto, o próximo agradecimento, vai a minha fámilia, que sempre me apoiou e ajudou quando possível a solucionar os problemas que surgiram durante o projeto, principalmente ao meu pai Marcos Cavalli e meu irmão Marcelo Ricardo Horst Cavalli, que me auxiliaram a realizar testes no projeto e a Cristiane Horst, que me auxiliou com a correção da monografia.

Também gostaria de agradecer aos amigos Wellingthon Mileo Junior e Felipe Douglas Makara, que me auxiliaram a resolver e investigar inúmeros problemas com *hardware* que ocorreram durante o decorrer do desenvolvimento e ao Eduardo Henrique Fideles Ribeiro por me auxiliar a testar a placa com sinais em um ambiente adequado.

Por último mas não menos importante, gostaria de agradecer a minha namorada, Pâmela dos Reis Moraes pelo apoio moral e incrivel dedicação em me auxiliar onde fosse possível e me manter sempre motivado para desenvolver este projeto.

"É melhor tentar e falhar, que preocupar-se e ver a vida passar.
É melhor tentar, ainda que em vão que sentar-se, fazendo nada até o final.
Eu prefiro na chuva caminhar, que em dias frios em casa me esconder.
Prefiro ser feliz embora louco, que em conformidade viver."
(Martin Luther King)

RESUMO

O objetivo desta monografia é projetar um sistema capaz de realizar a captura de sinais eletroencefalográficos através de eletrodos de superfície posicionados na região craniana do indivíduo. Essa captura foi realizada utilizando 8 canais diferenciais, totalizando 16 eletrodos. Durante a monografia, é realizado um estudo sobre o formato dos sinais eletroencefalográficos, sobre alguns circuitos integrados projetados especificamente para o fim de capturar estes sinais que estão sendo revendidos no mercado, sobre o funcionamento dos eletrodos para esta função e sobre os processadores mais indicados para realizar este feito. Após realizar a revisão literária é apresentado o passo a passo realizado para projetar e confeccionar a placa de aquisição de dados juntamente com a solução de alguns problemas mecânicos, como realizar o conector para os eletrodos na placa confeccionada. Nesta parte também é levantada uma planilha de custos referentes aos componentes utilizados na placa e a confecção da mesma. Concluindo a parte de *hardware*, é iniciado os procedimentos para a construção de um *software* para gerenciar todo o projeto. A primeira parte implementada foi o SPI, para criar a interface com o ADS1298, que é o coração da placa de aquisição. Durante esta etapa, notou-se que a raspberry selecionada não conseguiria realizar a comunicação necessária, então a mesma foi retirada do projeto, sendo substituída por um microcontrolador da família atmega. Devido a esta alteração, uma interface entre a saída USB da placa de aquisição de dados e o computador foi acrescentada ao projeto de *software*, esta saída já estava prevista em *hardware*. Ao completar o caminho do dado até o computador, foi implementado um filtro FIR com janelamento por *blackman* de ordem 50 e com frequência de corte em 45 Hz, este filtro foi escolhido devido ao menor impacto que o mesmo oferece à amplitude do sinal e ao fato de um filtro FIR não alterar em fase o sinal. Para encerrar a fase de desenvolvimento, foi elaborada uma janela gráfica para a exibição dos sinais capturados para o usuário. Esta janela de visualização foi construída através da biblioteca matplotlib usando principalmente a implementação da classe animated para gerar janelas visuais que conseguem exibir os 8 canais obtidos com um pequeno atraso. A monografia é finalizada com a exposição dos resultados finais do trabalho, com os gráficos obtidos durante uma tentativa de captura bem sucedida realizada em um indivíduo. É demonstrado o posicionamento final dos eletrodos através de fotos, junto com conselhos sobre como obter uma melhor resolução dos sinais finais.

Palavras-chaves: Eletroencefalografia, SPI, EEG, ADS1298, Eletrodos, Filtros, Matplotlib.

ABSTRACT

The objective of this monograph is to design a system capable of capturing electroencephalographic signals through surface electrodes positioned on the individual's cranial region. This capture was performed using eight differential channels, totaling 16 electrodes. A study was made on the format of electroencephalographic signals, on some integrated circuits designed specifically for the purpose of capturing these signals that are being resold in the market, on the operation of the electrodes for this function and on the processors best suited to perform this task. After performing a literary review, this monograph presents the steps performed to design and build the data acquisition board along with solving some mechanical problems, such as making the connector for the electrodes on the fabricated plate. In this part, it was created a cost spreadsheet referring to the components and building of the board. After the hardware part, the procedures for building software to manage the entire project are started. The first implemented part was SPI, to interface with ADS1298, which is the heart of the acquisition board. During this implementation, it was noted that raspberry would not be able to make the necessary communication, so it was removed from the project, being replaced by an atmega family microcontroller. Due to this change, an interface between the USB port of the data acquisition board and the computer had to be added to the software project, the hardware USB port was already planned. By completing the data path to the computer, a 50-order blackman windowed FIR filter with a 45 Hz cut-off frequency was implemented. This filter was chosen because of its lower impact on signal amplitude and the fact that FIR filter does not phase change the signal. To finish the development phase, a graphical window was elaborated to display the captured signals to the user. This viewport was built through the matplotlib library mainly using the animated class implementation to generate visual windows that could display the eight channels obtained with a short delay. To conclude, the final results of the work are presented with the graphs obtained during a successful capture attempt made on an individual. The final positioning of the electrodes is demonstrated through photos, along with advice on how to achieve better resolution of the final signals.

Key-words: Electroencephalography, SPI, EEG, ADS1298, Electrodes, Filters, Matplotlib.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EEG	Eletroencafalograma
BCI	*Brain-Computer Interface*
TCC	Trabalho de conclusão de curso
ADC	*Analogic - Digital Converter*
CMRR	*Common Mode Rejection Ratio*
SPI	*Serial Peripheral Interface*
SPS	*Samples per second*
CI	Circuitos Integrados
USB	*Universal Serial Bus*
DSP	*Digital Signal Processor*
UART	*Universal Asynchrounous Receiver/Transmiter*
I2C	*Inter-Integrated Circuit*
GPIO	*General Purpose Input/Output*
BGA	*Ball Grid Array*
MOSI	*Master Output/Slave Input*
MISO	*Master Input/Slave Output*
SCLK	*Clock Signal*
SS/CS	*Slave Select/Chip Select*
FFT	*Fast Fourier Transform*
FIR	*Finite Impulse Response*
IIR	*Infinite Impulse Response*
MSB	*Most Significant Bit*
RLD	*Right Leg Driver*
CSV	*Comma Separated Values*

SUMÁRIO

1 INTRODUÇÃO

Entender como o cérebro realiza suas atividades e consegue comandar o corpo humano é algo que tem atraído diversos cientistas desde Leonardo da Vinci até os tempos atuais. Para auxiliar isto, atualmente existem *hardwares* dedicados à captação e conversão destes sinais a fim de joga-los para outras plataformas e possibilitar o melhor estudo do mesmo.

Com interesse em aprofundar o conhecimento nesta área por parte dos pesquisadores, esta monografia visa estudar as formas como as ondas cerebrais funcionam à medida que o mesmo é utilizado para diversas atividades comuns, a fim de que seja possível transformar algumas atitudes do dia a dia em comandos para dispositivos externos, como por exemplo computadores.

Para que isto seja possível, inicialmente será estudado qual a melhor forma para extrai-las e amplifica-las a fim de possibilitar o trabalho de forma correta com as ondas. Dentro deste estudo também está incluso o estudo de mercado a fim de selecionar os melhores equipamentos de *hardware* disponíveis no mesmo para tornar esta tarefa possível e menos complicada.

Após selecionar o *hardware* adequado para o protótipo, será projetada e montada uma placa para realizar esta tarefa. Esta deverá aplicar um filtro no sinal recebido em suas entradas e amplifica-lo para que o mesmo seja processado por um meio externo a mesma. Para este projeto, este processamento será limitado a separação e caracterização das principais ondas de eletroencefalograma (EEG).

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Projetar uma placa dedicada para a extração de sinais gerados no cérebro com oito canais de captação de dados. Este *hardware* será composto pela parte de obtenção de dados, formado pelos eletrodos que serão utilizados, um filtro de baixas frequências analógico, uma parte de amplificação do sinal, que será feita através de um *hardware* dedicado e a parte final para o processamento digital do mesmo será externo a mesma.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- caracterizar o sinal desejado na saída;
- projetar uma placa de *hardware* dedicado a aquisição de sinais de EEG;
- criar uma interface para exibição do sinal;
- extrair o sinal de EEG corretamente.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte forma: o Capítulo 3, apresenta subtópicos explicando as teorias necessárias para o correto entendimento desta tarefa. Além disso, neste capítulo, podem ser vistos os conceitos dos *hardwares* que serão utilizados ao longo do desenvolvimento, juntamente com uma breve explicação sobre os *softwares* utilizados para o desenvolvimento deste projeto. O Capítulo 4 irá abranger toda a parte de desenvolvimento do projeto, com as aplicações de todos os conceitos abordados no Capítulo 3. Os resultados desta parte do desenvolvimento do trabalho encontram-se no Capítulo 5. Por fim, no capitulo 6 estão as conclusões finais do projeto.

2 METODOLOGIA

2.1 DESCRIÇÃO DA MONOGRAFIA

Aquisitar os dados referentes a atividade cerebral de um paciente não é uma coisa trivial. Sendo assim, para tornar esta atividade possível, este trabalho foi dividido em cinco partes:

- Captar o sinal;
- Tratar o sinal analógico;
- Converter o sinal;
- Filtrar o sinal digital;
- Disponibilizar o sinal em uma interface amigável.

Para a parte de obtenção do sinal, serão utilizados eletrodos como meios de captura do mesmo. Estes eletrodos estarão fixados na cabeça do paciente com uma pasta especifica. A saída dos eletrodos estará conectada na placa de análise inicial do sinal.

Esta placa, construída pelo autor do trabalho, será responsável pela filtragem analógica, feita com filtros passivos, pela conversão do sinal analógico para digital e por transmitir este sinal para alguma interface de processamento.

A parte final irá implementar filtros digitais em cima do sinal, a fim de melhorar a visualização final do mesmo.

Por fim, esta parte também será responsável pela exibição deste sinal em uma interface amigável ao usuário que será montada ao decorrer do trabalho.

2.2 FLUXOGRAMA

O ponto inicial do projeto é a correta fixação dos eletrodos na cabeça do paciente. Isto será feito utilizando um creme adesivo e condutivo especifico para exames de eletroencéfalograma (EEG). Estes eletrodos estarão conectados na entrada da placa projetada, que possuirá filtros passa-baixa analógicos configurados com o valor comercial mais próximo possível de 150 Hz.

Após este filtro inicial, o sinal captado será amplificado através de um amplificador operacional diferencial, que estará ligado a dois eletrodos distintos, formando um canal. O sinal é então transmitido para um conversor AD, ainda na placa inicial.

O sinal digital é então passado para a parte responsável pelo processamento através do protocolo de comunicação *Serial Peripheral Interface* (SPI), poderá ser utilizado qualquer processador que tenha capacidade para realizar esta comunicação. Uma vez dentro da parte de processamento, este sinal será filtrado digitalmente e então exibido para o usuário final.

Na figura 1 é exposto o fluxo do projeto que foi descrito nos paragrafos anteriores.

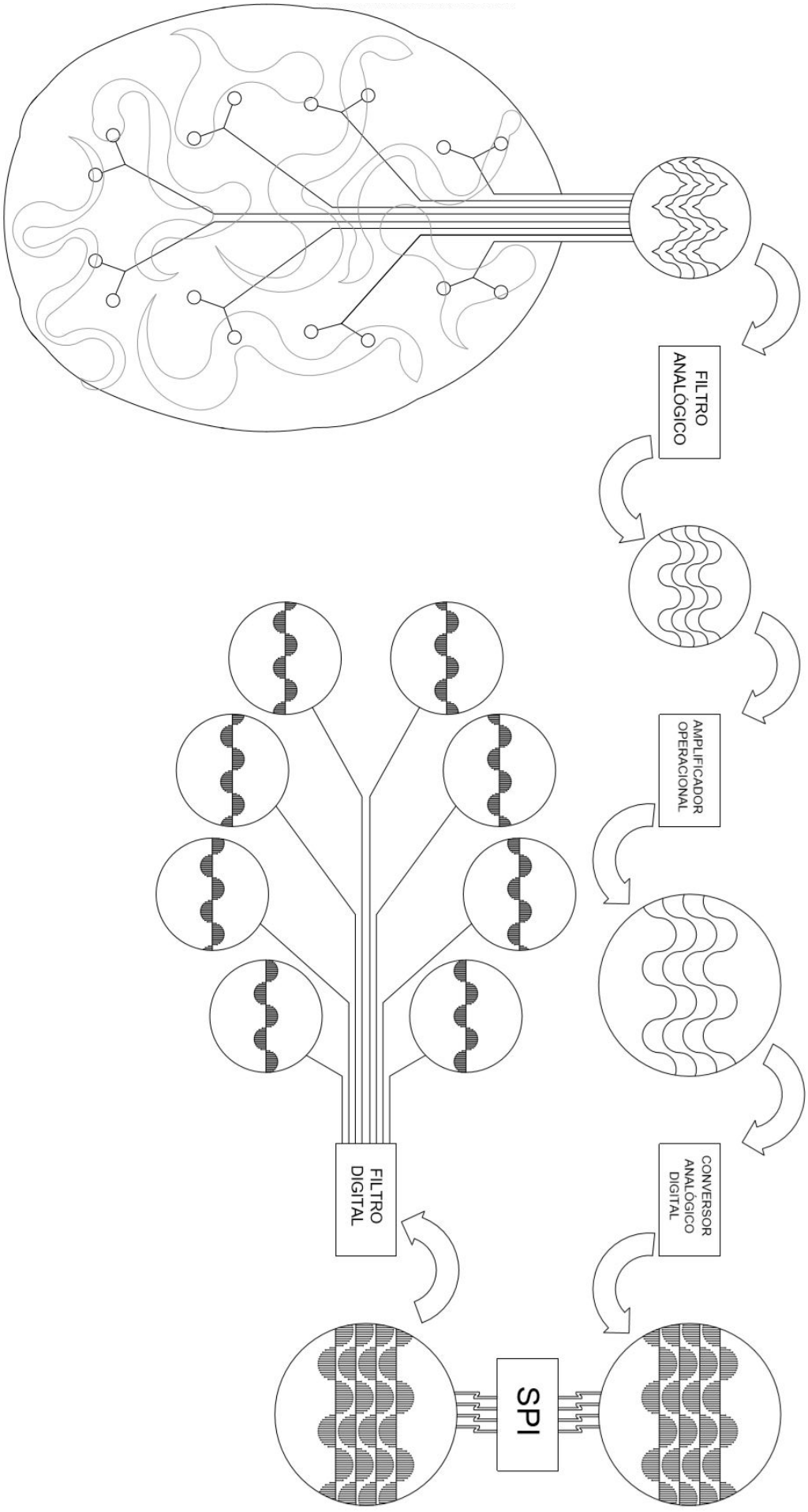

Figura 1 – Fluxograma do trabalho. – Fonte: Autor.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 CARACTERÍSTICAS DO SINAL

Nesta seção da monografia estão descritas as principais características sobre o sistema nervoso central do corpo humano, as quais são necessárias para o correto entendimento desta tarefa.

3.1.1 Sistema nervoso

O sistema nervoso do corpo humano é responsável fisiológicamente por três funções, captar mudanças nos meios internos e externos ao próprio, através de estimulos sensoriais, processar estas mudanças captadas e, ao final, responder o evento ocorrido com algum estímulo motor que seja apropriado para o ocorrido. Isto ocorre através de milhões de sinais oriundos e trocados por diversas partes do corpo (GUYTON, 1992).

Estes sinais elétricos que percorrem o corpo são difundidos pelo mesmo através de diversos neurônios que, juntos, formam algumas redes neurais.

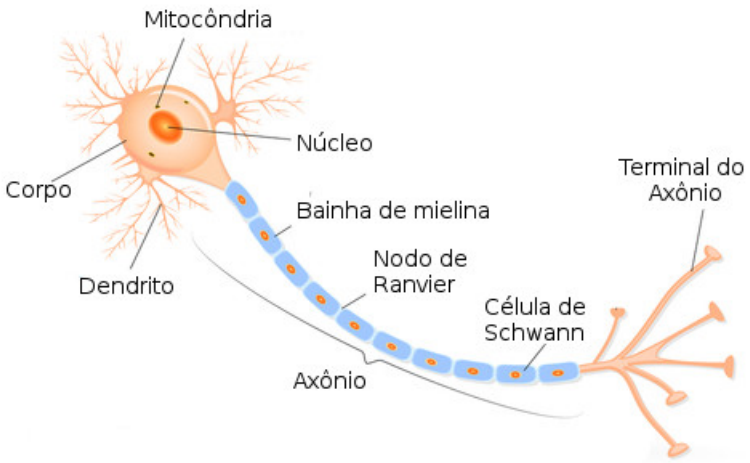

Figura 2 – Neurônio – Fonte: (INFOESCOLA, 2007)

Como pode ser observado na figura 2, um neurônio pode ser dividido em diversas partes. Indo de baixo para cima, a primeira parte seria o terminal do axônio, responsável por transmitir o pulso elétrico para o próximo neurônio, está localizada na extremidade do axônio e forma a maior fração do corpo do neurônio. No trecho superior do mesmo, é possível ver os dendritos, que é a responsável por captar o pulso elétrico que será transmitido através dos outros neurônios (POERSCH, 2004).

Entre os terminais do axônio de um neurônio e os dendritos de outro é onde ocorre a sinapse, que é a reação química responsável pela passagem da informação entre os neurônios. Desta forma, a informação transita entre o cérebro e algum ponto do corpo (POERSCH, 2004).

3.1.2 Sinais eletroencefalográficos

As transmissões dessas informações podem variar em quesitos de frequência e amplitude, de acordo com a sua origem, destino e tipo de informação transmitida. Dentre as ondas geradas no cérebro é possível notar frequências alternando desde 1 até 150Hz, sendo que as principais ondas utilizadas para análise estão na faixa de 1 até 30 Hz (BOTUCATU, 2015). Dentro desta faixa principal, as ondas podem ser separadas em quatro grupos:

- Ondas betas: variam entre 14 e 30 Hz. Normalmente a existência desta onda demonstra que o indivíduo está em estado de concentração (BOTUCATU, 2015);
- Ondas alfas: variam entre 09 e 13 Hz. Normalmente a existência desta onda mostra que o indivíduo se encontra em estado de relaxamento (BOTUCATU, 2015);
- Ondas thetas: variam entre 04 e 08 Hz. Normalmente a existência desta onda demonstra que o indivíduo está em estado de meditação profunda ou sono raso (BOTUCATU, 2015);
- Ondas deltas: variam entre 01 e 03 Hz. Normalmente a existência desta onda mostra que o indivíduo está em estado de sono profundo (BOTUCATU, 2015).

Na figura 3 pode ser observado um exemplo de cada tipo de onda.

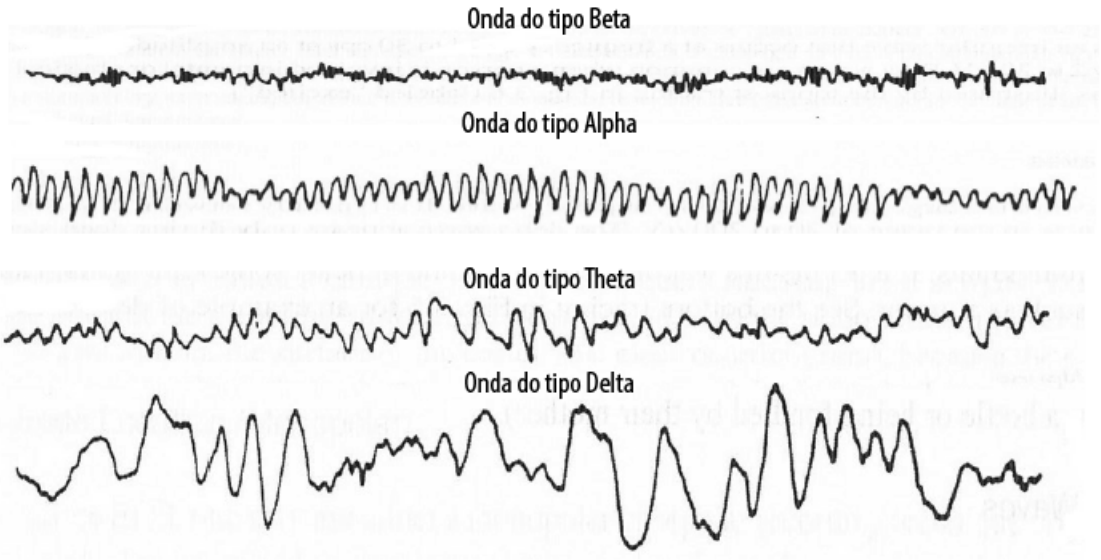

Figura 3 – Exemplo de ondas cerebrais – Fonte: (BOTUCATU, 2015)

3.2 ELETRODOS

3.2.1 Características

É um componente cujo objetivo é realizar a transferência de elétrons entre o circuito e o meio no qual o mesmo está inserido (ALCANTARA, 2006).

É muito dificil extrair bons registros de EEG sem eletrodos de qualidade, pois estes são a interface entre o couro cabeludo (local de onde serão extraidas as atividades elétricas cerebrais) e os amplificadores do eletroencefalografo (CABOCLO, 2013).

Esses eletrodos podem ser passivos, ativos ou secos.

Eletrodos passivos são aplicados ao couro cabeludo utilizando um gel ou creme adesivo e condutivo, geralmente estes eletrodos são banhados com cloreto de prata (Ag-AgCl) devido a suas propriedades condutivas. Já os eletrodos secos podem ser aplicados sem o uso de pasta e/ou gel condutor. Por sua vez, os eletrodos ativos possuem circuitos que realizam o casamento de impedância entre o couro cabeludo e o eletrodo, para isso se faz necessário uma fonte externa de energia para adquirir o sinal. A relação sinal-ruído desses eletrodos é melhor quando comparado aos modelos convencionais. Eletrodos ativos podem ser projetados tanto para serem utilizados secos quanto com gel e pasta condutora (CABOCLO, 2013).

Os eletrodos atuam como transdutores moduladores capacitivos não polarizáveis, isto significa que, estes eletrodos conseguem transmitir o sinal recebido por uma das extremidades sem modificar as propriedades do mesmo, não sendo necessário gastar alguma energia para que funcionem (WEBSTER, 2010).

Na figura 4 pode ser visto uma representação do circuito formado desde a geração do sinal pelos neurônios até a captação dos mesmos pelos eletrodos.

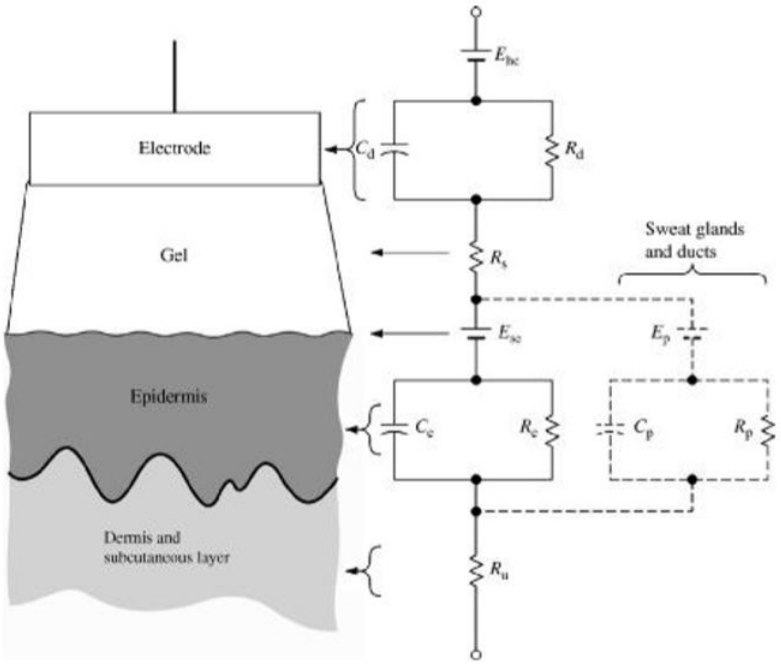

Figura 4 – Circuito equivalente dos eletrodos. Fonte: (WEBSTER, 2010).

Nota-se que, desde sua geração no neurônio, o sinal deverá passar por algumas camadas, como o crânio e a epiderme até chegar a superficie, onde irá passar pelo gel e por fim chegará ao eletrodo. Estas camadas reduzem significativamente a amplitude da tensão esperada para estes impulsos nervosos, chegando a superficie crâniana com uma tensão entre

20 e 200μ V, porém, na região do lóbulo da orelha estas ondas podem chegar até 10m V (WEBSTER, 2010).

Para este projeto serão utilizados eletrodos passivos de EEG banhados a ouro, pois, com a pasta específica para esta função, eles garantem uma boa condutividade e também serão corretamente fixados no couro cabeludo do indivíduo. O cenário ideal seria a utilização de uma touca de EEG, porém a mesma não foi encontrada no mercado.

3.2.2 Posicionamento dos eletrodos

Para obtenção do sinal eletroencefalográfico normalmente se utiliza eletrodos, alguns destes necessitam o auxílio de um gel condutor para manter contato com a pele, conforme explicado previamente. Existe uma proposta internacional feita nos anos 50 do posicionamento dos eletrodos para assegurar a padronização de pesquisas cientificas. Estas foram escolhidas estrategicamente em relação as diversas regiões cerebrais. Assim, independente da formação do crânio, as posições dos eletrodos se mantem para diferentes biotipos distintos (SANEI, 2007).

De acordo com a proposta realizada, os eletrodos são posicionados de forma simétrica, conforme figura 5 (SANEI, 2007).

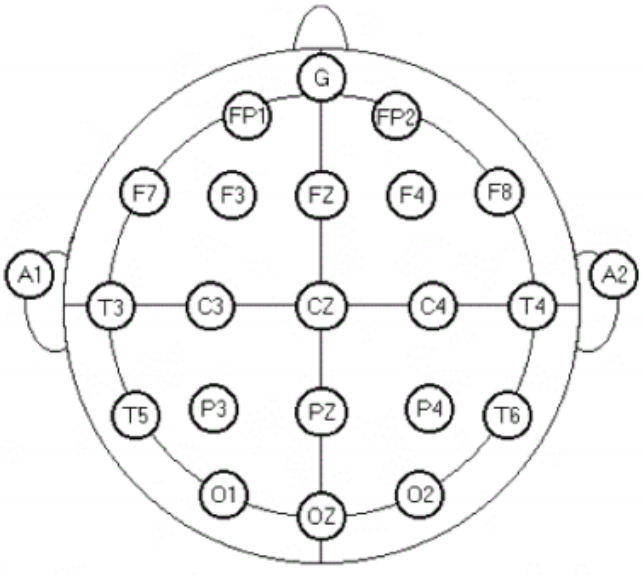

Figura 5 – Representação das localizações dos eletrodos no couro cabeludo. Fonte: (SANEI, 2007).

Com o aumento da quantidade de eletrodos, tem-se a melhora da coleta dos sinais possibilitando uma melhor identificação dos potenciais elétricos pois existirão uma maior quantidade de sinais para serem analisados.

Para futuros trabalhos, as possíveis regiões de maior interesse podem ser:

- o lobo frontal - responsável pelo movimento e pensamento;
- o lobo parietal - responsável por localização espacial, compreensão das relações espaciais.

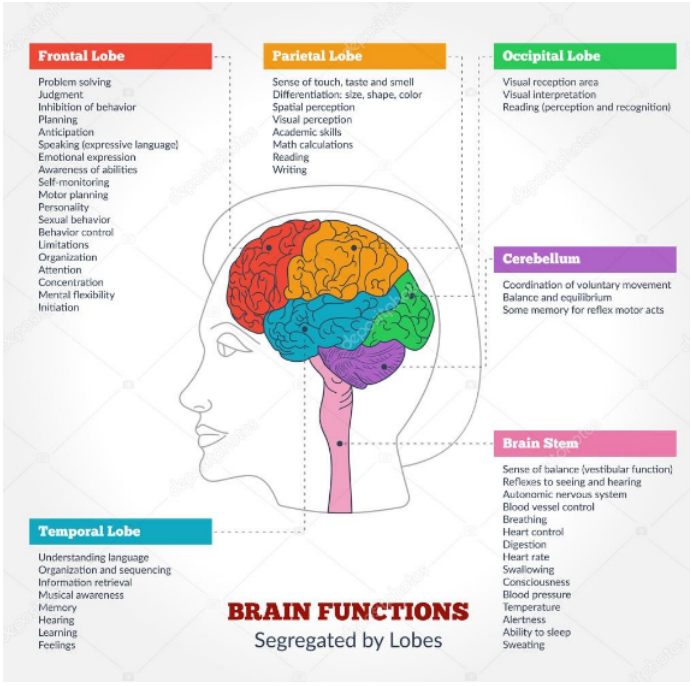

Figura 6 – Representação das funções do cerebro. Fonte: (SUNSHINE_ART, 2015).

As diferentes regiões nas quais o cerébro é mais comumente separado podem ser observadas na figura 6. Cada região possuí uma função distinta e podem ser usadas para observar diferentes fatores.

3.3 FILTROS

Um filtro pode ser classificado de duas maneiras diferentes, quanto a tecnologia de construção utilizada neles ou quanto ao modelo de resposta para diferentes estímulos dos mesmos.

Quanto a tecnologia utilizada, um filtro pode ser do tipo passivo, ativo ou digital.

Quando um filtro é do tipo passivo, isto significa que o mesmo foi construído apenas com elementos passívos do circuito, que não são capazes de produzir energia, ou seja, um filtro passivo é composto apenas de resistores, capacitores e/ou indutores. Já um filtro ativo mescla em sua composição elementos passivos com elementos ativos, isto é, além de utilizar os mesmos componentes do filtro passivo, este filtro também faz uso de componentes ativos, como amplificadores operacionais e transistores. Também existem os filtros digitais, implementados através de programas em dispositivos programáveis. (ARNDT, 2016)

O tipo de resposta produzida pelo filtro pode ser classificada em:

- Filtro passa-baixa;
- Filtro passa-alta;
- Filtro passa-faixa;
- Filtro rejeita-faixa;

O filtro passa-baixa permite que ondas que possuem uma frequência menor do que a especificada no projeto do mesmo passem por ele. Já o filtro passa-alta possuí um comportamento inverso, permitindo que todas as ondas de frequência maior que a qual ele foi projetado para cortar passem. Há ainda o filtro passa-faixa, que permite que apenas sinais que estão entre uma faixa de frequência se propaguem por ele. De maneira oposta, existem também os filtros rejeita-faixa que recusam apenas uma faixa especifica de frequência. Na figura 7 é possível observar as respostas dos diferentes tipos de filtros citados.

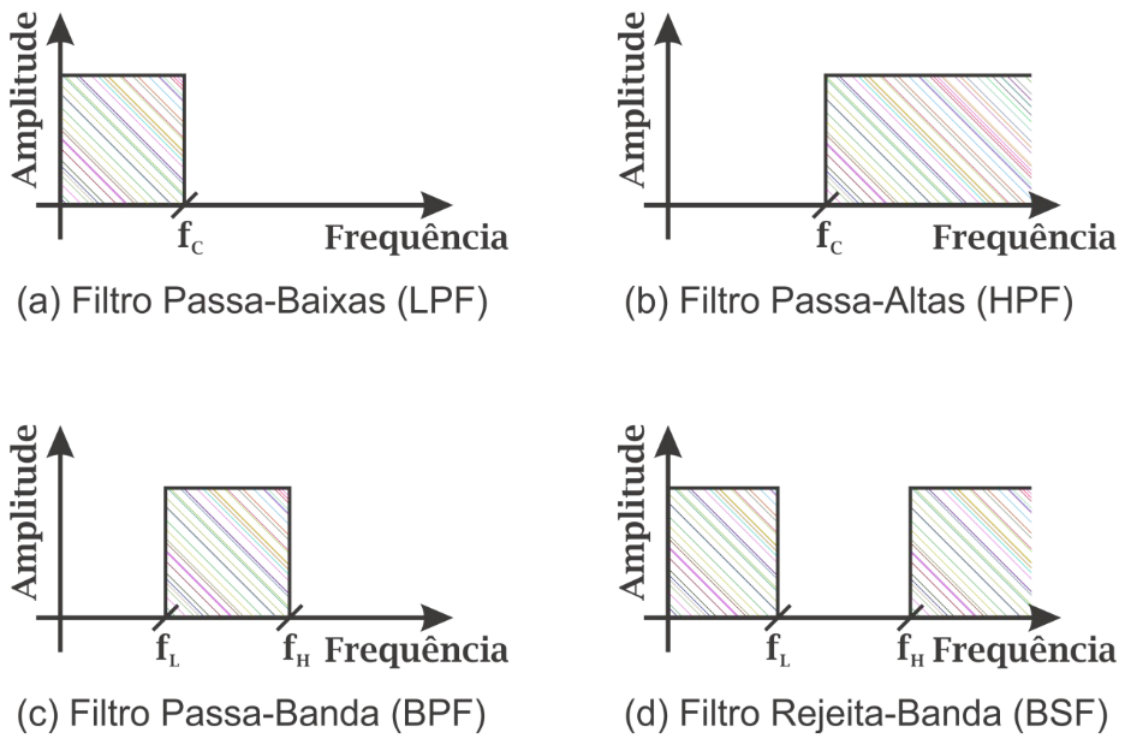

Figura 7 – Exemplo de filtros ideais. Fonte: (ARNDT, 2016).

Dentre os filtros digitais, há ainda algumas separações baseadas em sua forma de funcionamento, as principais são:

- Filtros *Finite Impulse Response* (FIR);
- Filtros *Infinite Impulse Response* (IIR);
- Filtros adaptativos.

Os filtros FIR são filtros que são normalmente projetados para atuarem sobre uma faixa de frequência, tendo sua resposta nula após um período de tempo finito, são estaveis, possuem fase linear e permitem o uso de convolução para o processo de filtragem. Estes filtros não utilizam realimentação de sinais para seu funcionamento normal. Exemplos destes filtros são as janelas de *Hanning*, *Hamming* e *Blackman*. Filtros IIR possuem sua resposta ao impulso infinita, não são lineares e dependem de realimentação. Exemplos de técnicas destes filtros são *Butterworth* e *Chebyshev*. Por ultimo, existem os filtros adaptativos, que adaptam seu

funcionamento baseado nos valores anteriores ao mesmo e a alguns parametros. Exemplos de filtros adaptativos são a média móvel, o filtro *Least mean squares*, redes neurais etc. (PAVAN, 2011).

3.4 *HARDWARE*

Um dos primeiros problemas encontrados para este projeto foi escolher qual *hardware* utilizar para a montagem da placa de aquisição de sinais de EEG. O dilema enfrentado foi principalmente a escolha entre montar o circuito amplificador do sinal ou utilizar um circuito integrado (CI), que fosse comercializado por uma empresa do mercado e que garantisse tudo o que fosse necessário para trabalhar com sinais na magnitude de um sinal de EEG.

Após pesquisar sobre, foram selecionados três dispositivos da fabricante *Texas Instruments* que são capazes de realizar a tarefa necessária, são eles:

- ADS1198;
- ADS1298;
- ADS1299.

Para poder selecionar o CI mais adequado para o projeto, serão vistas as principais características destes *hardwares* nas sessões 3.4.1, 3.4.2 e 3.4.3.

3.4.1 ADS1198

O ADS1198 faz parte de uma linha de CIs projetados para realizar a amostragem simultânea de diversos canais, podendo variar entre 4, 6 ou 8, já integrados com amplificadores de ganhos programáveis e um oscilador interno (TEXAS INSTRUMENTS, 2019a).

As principais características deste componente para o projeto em que será desenvolvido são:

- 8 amplificadores diferenciais;
- 8 ADCs de 16 bits;
- Taxa de transferência de dados: 125SPS até 8kSPS;
- CMRR: –105dB;
- Ganho programável: 1, 2, 3, 4, 6, 8, ou 12;
- Interface de saída de dados SPI.

Este CI possui a dimensão externa de 1,2 cm x 1,2 cm. Na figura 8 se encontra o diagrama funcional do mesmo fornecido pelo fabricante.

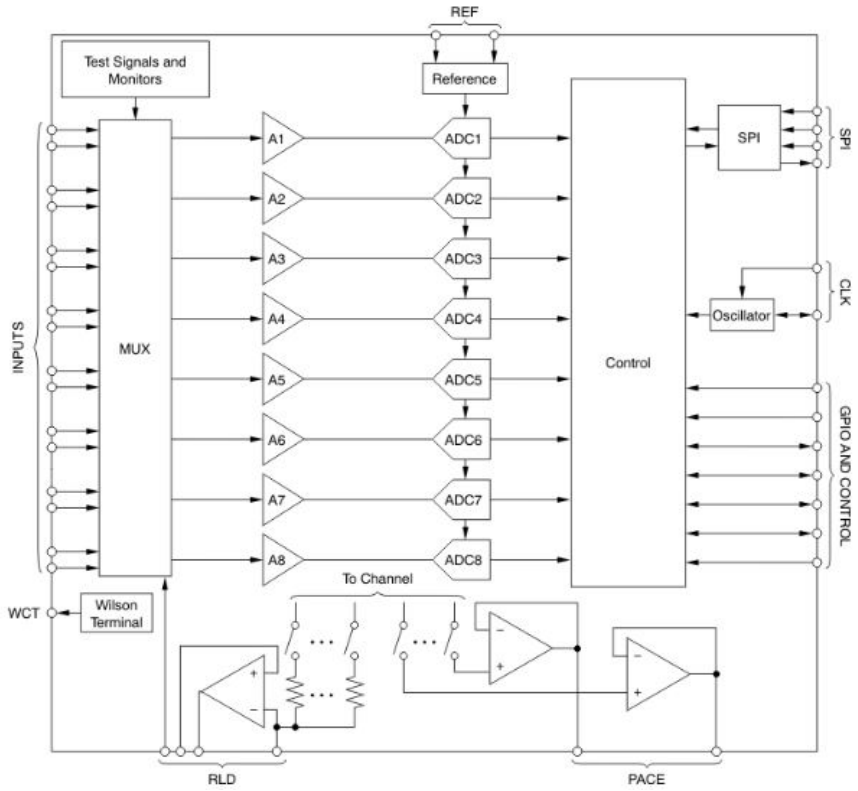

Figura 8 – Diagrama funcional do ADS1198 – Fonte: (TEXAS INSTRUMENTS, 2019a)

3.4.2 ADS1298

O ADS1298 faz parte de uma linha de CIs projetados para realizar a amostragem simultânea de diversos canais, podendo variar entre 4, 6 ou 8, já integrados com amplificadores de ganhos programaveis e um oscilador interno (TEXAS INSTRUMENTS, 2019b).

As principais características deste componente para o projeto em questão são:

- 8 amplificadores diferenciais;
- 8 ADCs de 24 bits;
- Taxa de transferência de dados: 250 SPS até 32 kSPS;
- CMRR: –115dB;
- Ganho programável: 1, 2, 3, 4, 6, 8, ou 12;
- Interface de saída de dados SPI;
- Detecção de eletrodo desconectado;
- Terminal central de Wilson;
- Amplificador Integrado de Perna Direita.

Este CI possui a dimensão externa de 1,2 cm x 1,2 cm. Na figura 9 se encontra o diagrama funcional fornecido pelo fabricante.

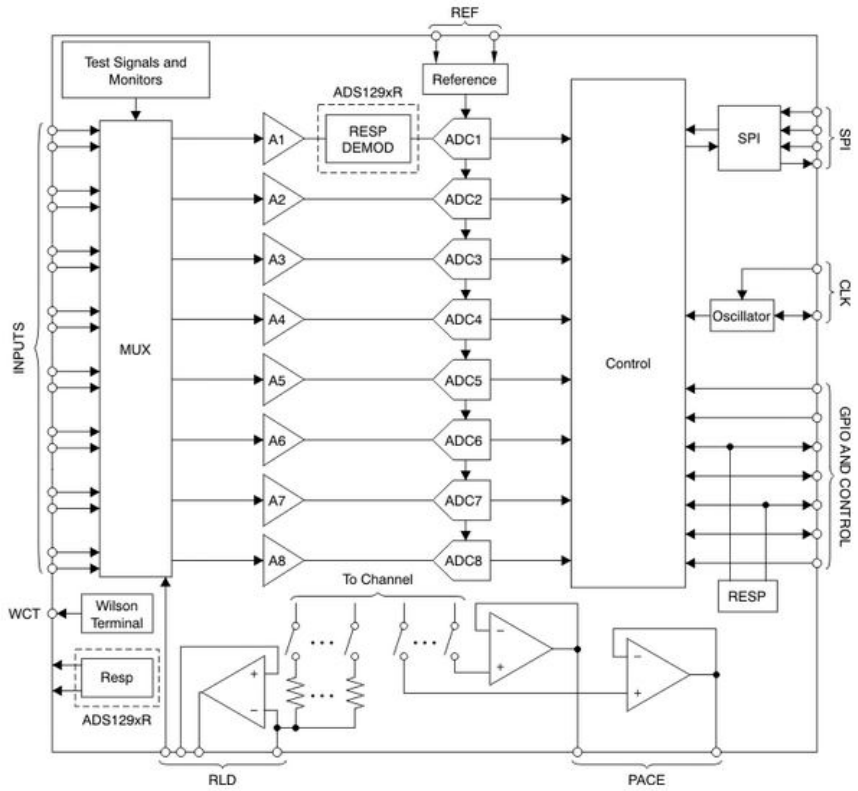

Figura 9 – Diagrama funcional do ADS1298 – Fonte: (TEXAS INSTRUMENTS, 2019b)

3.4.3 ADS1299

O ADS1299 faz parte de uma linha de *chips* projetados para realizar a amostragem simultânea de diversos canais, podendo variar entre 4, 6 ou 8, já integrados com amplificadores de ganhos programaveis e um oscilador de sinais interno (TEXAS INSTRUMENTS, 2019c).

As principais características deste componente para o projeto em questão são:

- 8 amplificadores diferenciais;
- 8 ADCs de 24 bits;
- Taxa de transferência de dados: 250 SPS até 16 kSPS;
- CMRR: –110dB;
- Ganho programável: 1, 2, 3, 4, 6, 8, 12 ou 24;
- Interface de saída de dados SPI;
- Detecção de eletrodo desconectado;
- Terminal central de Wilson;
- Amplificador Integrado de Perna Direita.

Este CI possui a dimensão externa de 1,2 cm x 1,2 cm. Na figura 10 se encontra o diagrama funcional do mesmo fornecido pelo fabricante.

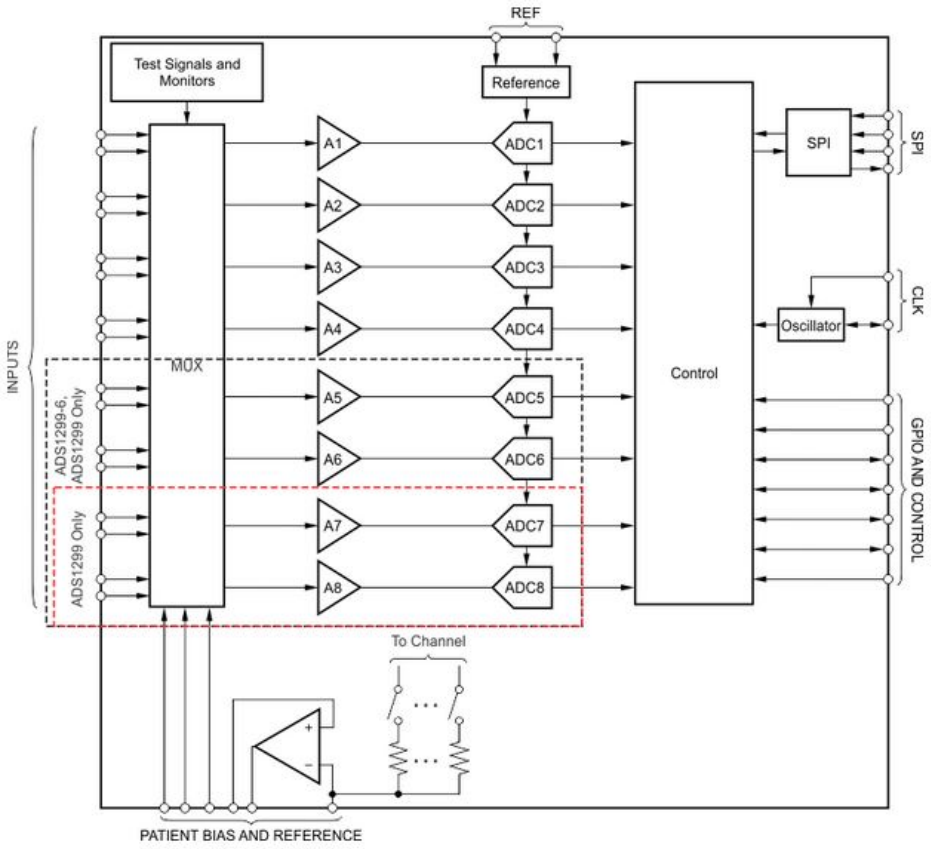

Figura 10 – Diagrama funcional do ADS1299 – Fonte: (TEXAS INSTRUMENTS, 2019c)

3.5 PROCESSAMENTO

O processamento dos sinais de 8 canais de um EEG não é uma tarefa para qualquer processador. Sendo assim, uma das preocupações com este trabalho foi encontrar um processador que fosse capaz de processar, filtrar e separar cada uma das ondas de cada um dos canais.

Ao mesmo tempo, este processador teria que possuir uma boa conectividade e versatilidade, a fim de possibilitar futuros trabalhos e não se tornar um problema para os mesmos.

Tendo em mente estas questões, existem 3 possíveis frentes a serem seguidas a fim de solucionar este problema, são elas:

- utilizar um microcomputador;
- utilizar um microcontrolador;
- construir uma placa com um Processador de Sinal Digital (DSP) para o processamento do mesmo.

Dentre as soluções pesquisadas, os principais representantes de cada linha de solução levantada encontrados no mercado foram:

3.5.1 Raspberry Pi

O *Raspberry Pi* 3 modelo B é um microcomputador muito versátil e difundido no meio academico. Suas principais características são (RASPBERRY, 2018):

- processador *Quad-Core* de 1,2 GHz com CPU de 64*bits*;
- 1 GB de memória *RAM*;
- comunicação *Wi-Fi* e *Bluetooth* embarcada;
- 4 Portas USB;
- uma porta *HDMI* e uma saída de áudio formato P2;
- leitor de cartão microSD;
- pinos de saída para o protocolo SPI.

Outra grande característica dos microcomputadores da linha *Raspberry Pi* é o fato dos mesmos permitirem que um sistema operacional seja embarcado em sua memória. Isto garante que diversas linguagens de programação possam ser compiladas e rodadas sobre o mesmo, criando uma gama de possibilidades para construção de *scripts* e futuras melhorias. O microcomputador pode ser visto na figura 11.

Figura 11 – Raspberry Pi 3 Modelo B – Fonte: (RASPBERRY, 2018)

3.5.2 Texas TM4C123G *LaunchPad*

O microcontrolador TM4C123G *LaunchPad* pertence a uma linha da fabricante Texas Instruments que opera em cima de um *chipset* tiva, muito utilizado para o processamento de sinais. Isto se dá principalmente devido ao seu processador de 32 *bits* e seu duplo ADC de 12 *bits* cada. Suas principais caracteristicas são (TEXAS INSTRUMENTS, 2019d):

- 80MHz 32-bit CPU;
- 256kB Flash, 32KB SRAM, 2kB EEPROM;
- USB 2.0 Host/Device/OTG;
- Duplo ADC 12-bit 2MSPS;

- 8 pinos para UART;
- 6 pinos para I2C;
- 4 pinos para SPI;

Este microcontrolador, apesar de simples, é capaz de operar com o sinal necessário, logo o mesmo se torna uma possibilidade para este projeto. O microcontrolador pode ser visto na figura 12.

Figura 12 – Texas TM4C123G *LaunchPad* – Fonte: (TEXAS INSTRUMENTS, 2019d)

3.5.3 DSP - TMS320C5535

O TMS320C5535 é um processador especializado na operação com sinais digitais. Construído pela fabricante Texas Instruments, este processador possuí um tempo de instrução de até 10ns, tendo como frequência máxima de clock 100 MHz. Suas principais caracteristicas são (TEXAS INSTRUMENTS, 2019e):

- 20-10ns tempo do ciclo de instrução;
- 50-100MHz de frequência de clock;
- 320kB de RAM;
- Três 32-Bit timers;
- Conectividades: UART, SPI, I2C;
- 32 GPIO Pins;
- Tensão de alimentação de 3,3V;

Este DSP possuí um encapsulamento do tipo *Ball Grid Array* (BGA), o que pode dificultar o processo de montagem de uma placa dedicada para o mesmo, pois não é facil achar um local para realizar a solda de componentes com este tipo de engate. Sendo assim,

por mais que este componente possua uma capacidade de processamento altamente elevada, o mesmo provavelmente não será utilizado durante o decorrer do projeto.

Figura 13 – Texas TMS320C5535 – Fonte: (TEXAS INSTRUMENTS, 2019e)

3.6 *SOFTWARE*

Dentro deste projeto está prevista uma parte de *software* para gerenciar a comunicação entre as partes de obtenção do sinal e processamento do mesmo, implementar um filtro digital e montar uma interface amigavel para o usuário.

As sessões abaixo descrevem o funcionamento geral de cada uma das partes abordadas na seção 4.

A principio, será utilizada a linguagem de programação python para os itens que possam ser desenvolvidos em cima de um sistema operacional, devido a simplicidade que a mesma oferece aliada a um forte poder de processamento e a linguagem nativa do arduino para caso seja necessário a utilização do mesmo para a comunicação SPI.

3.6.1 Protocolo de comunicação SPI

Este protocolo de comunicação é amplamente suportado por diversos dispositivos, variando desde cartões SD até transmissores/receptores *wireless* (CIRCUITS BASICS, 2016).

Este procotolo funciona com um esquema de *master-slave* para o funcionamento da comunicação. Nele, o *master* gera um sinal de *clock* que será responsável pela sincronização do emissor e receptor dos dados, a velocidade de transmissão também varia em função deste *clock* (CIRCUITS BASICS, 2016).

No protocolo SPI, são necessárias quatro vias de comunicação, uma para a saída de dados do *master* para o *slave*, que seria a *Master Output/Slave Input* (MISO), outra para a saída de dados do *slave* indo sentido *master*, nomeada de *Master Input/Slave Output* (MISO), a terceira via de comunicação é para o sinal de *clock* (SCLK) e a última serve para a seleção do *slave* com o qual o *master* está trocando dados no momento, nomeade de *Slave Select/Chip Select* (SS/CS).

Na figura 14 pode ser vista a configuração deste protocolo, com o sentido do fluxo dos dados (CIRCUITS BASICS, 2016).

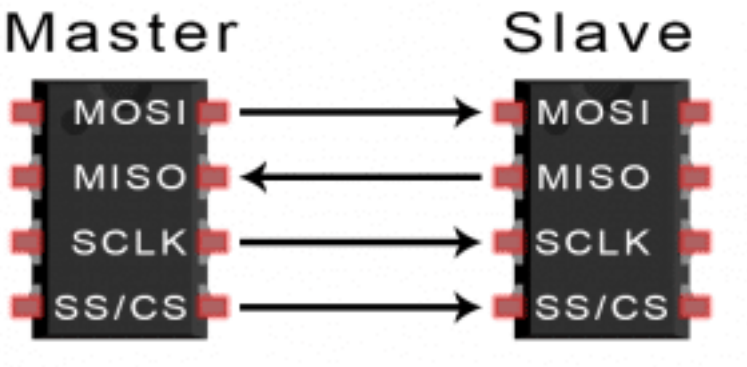

Figura 14 – SPI - Exemplo de implantação. – Fonte: (CIRCUITS BASICS, 2016)

As principais vantagens deste protocolo são:

- Não há *bits* indicando o começo e o fim da mensagem, isto permite um fluxo contínuo na transmissão de dados;
- Não há um sistema complicado de endereçamento como outros protocolos;
- Alta velocidade de transmissão de dados, chegando a 10 Mbps;
- Linhas de comunicação dedicadas para o MISO e o MOSI, permitindo que ocorra comunicação em paralelo entre ambas as placas.

Por sua vez, as principais desvantagens deste protocolo são:

- Utiliza quatro vias de transmissão;
- Não há um pacote de confirmação para o recebimento da mensagem;
- Não há suporte nativo para detecção de erros na transmissão;
- Este protocolo permite apenas um mestre na implementação.

Como ambas as placas possuem suporte nativo para este protocolo de comunicação e as desvantagens do protocolo não deverão causar nenhum problema para este projeto, este foi o protocolo escolhido para realizar a comunicação entre a aquisição e o processamento de dados.

Para o projeto em questão, a uma primeira tentativa a *Raspberry* deverá funcionar como *master* e a placa responsável pela aquisição de dados terá o papel de *slave*. Caso ocorra alguma falha, será tentado substituir a placa *Raspberry* por uma que não possua um sistema operacional não *real-time* embarcado.

3.6.2 Implementação do protocolo

O primeiro foco do *software* será realizar a interface entre a placa construida e o microcomputador escolhido para realizar a tarefa.

Para um primeiro teste dentro do ambiente da *Raspberry* será utilizada a biblioteca *gpiozero*, construida em python, cuja aplicação é criar uma interface entre os pinos de *General*

Purpose Input/Output (GPIO) da Raspberry Pi e o *software* que está sendo construído. Para este caso, será utilizado para interfacear os pinos do protocolo de SPI de saída da placa construída com as GPIO dedicadas a esta tarefa no microcomputador em questão.

A implementação desta parte do programa ocorre de maneira simples e será mostrada mais para frente como realizar a correta configuração da mesma.

Caso o ambiente da *Raspberry* falhe, será utilizada a biblioteca SPI do Arduino para o mesmo fim.

3.6.3 Implementação dos filtros digitais

Para a implementação dos filtros digitais existem duas possibilidades, um delas é projetar o filtro e implementa-lo no código e a outra é a utilização de uma biblioteca que já é capaz de implementar estes filtros no python.

Uma biblioteca que já implementa este filtro é a biblioteca SciPy, que é *open source* e implementa inumeras técnicas para tratamento de sinais de processamento de dados. Esta biblioteca utiliza como base a biblioteca NumPy, que tem sua base em C, e também implementa alguns algoritmos em C, o que resulta em um programa simples aos olhos do usuário porém muito eficiente computacionalmente (PYSCIENCE-BRASIL, 2010).

Algumas das aplicações desta biblioteca são (PYSCIENCE-BRASIL, 2010):

- Estatísticas;
- Otimização;
- Integração numérica;
- Processamento de sinais e imagens;
- Solução de equações diferenciais;
- Funções especiais, como Bessel etc.;
- Polinômios.

4 DESENVOLVIMENTO

4.1 PROJETO

4.1.1 Esquemáticos

O desenvolvimento do projeto do sistema de captação dos dados se iniciou pela decisão do componente central do mesmo, responsavel pela amplificação do sinal e pela conversão do mesmo. Dos componentes revisados em 3.5, foi optado por utilizar o CI ADS1298, explicado no item 3.4.2.

Foi optado por este componente devido aos 8 ADCs internos de 24bits, aliado ao seu alto valor de CMRR e a possibilidade de integrar os dados de saída com o processador através da interface SPI, que garante estabilidade e permite simplificar a parte de conexão desde que a placa escolhida para o processamento contenha o mesmo conector.

Após esta decisão, a etapa de projeto da placa foi iniciada. Como era esperado uma complexidade alta para este projeto, foi escolhido realizar a confecção da placa por parte de uma empresa profissional.

Para o mesmo, foi selecionado uma empresa chinesa, a JLCPCB, que possui o ambiente de desenvolvimento de placas integrado da empresa EasyEDA e também a revendedora de componentes eletrônicos LCSC. Esta empresa foi selecionada devido à facilitar a integração entre estas partes.

A primeira etapa projetada foram os filtros analógicos pelos quais os sinais passariam. Como se trata de um sinal para EEG, este filtro deve ser formado exclusivamente de componentes passivos, para não afetar o sinal, sendo assim, foi confeccionado apenas com capacitores e resistores. Este filtro está na figura 15. Os valores dos componentes foram calculados para ser um filtro passa-baixa que possui sua faixa de corte de frequência em 150 Hz.

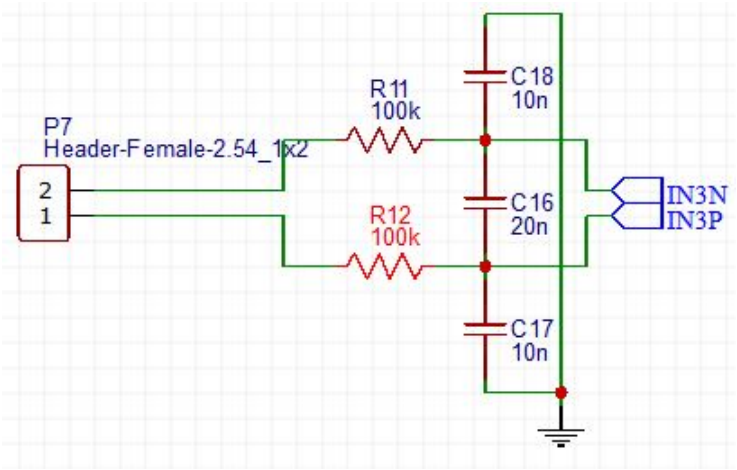

Figura 15 – Esquemático dos filtros para cada eletrodo – Fonte: Autor.

A próxima etapa projetada foi a fonte que irá regular a tensão dos componentes da placa. O esquemático criado para este circuito se encontra na figura 16. As tensões de saída

foram calculadas para alimentar o ADS1298, conforme especificado no site da fabricante. Esta tensão analógica foi mantida em 5 V, que estaria entre a faixa de 2,7 V e 5,25 V e a tensão digital foi planejada para 3,3 V, que estaria entre a faixa de 1,65 V e 3,6 V, conforme especificado pelo fabricante (TEXAS INSTRUMENTS, 2019b).

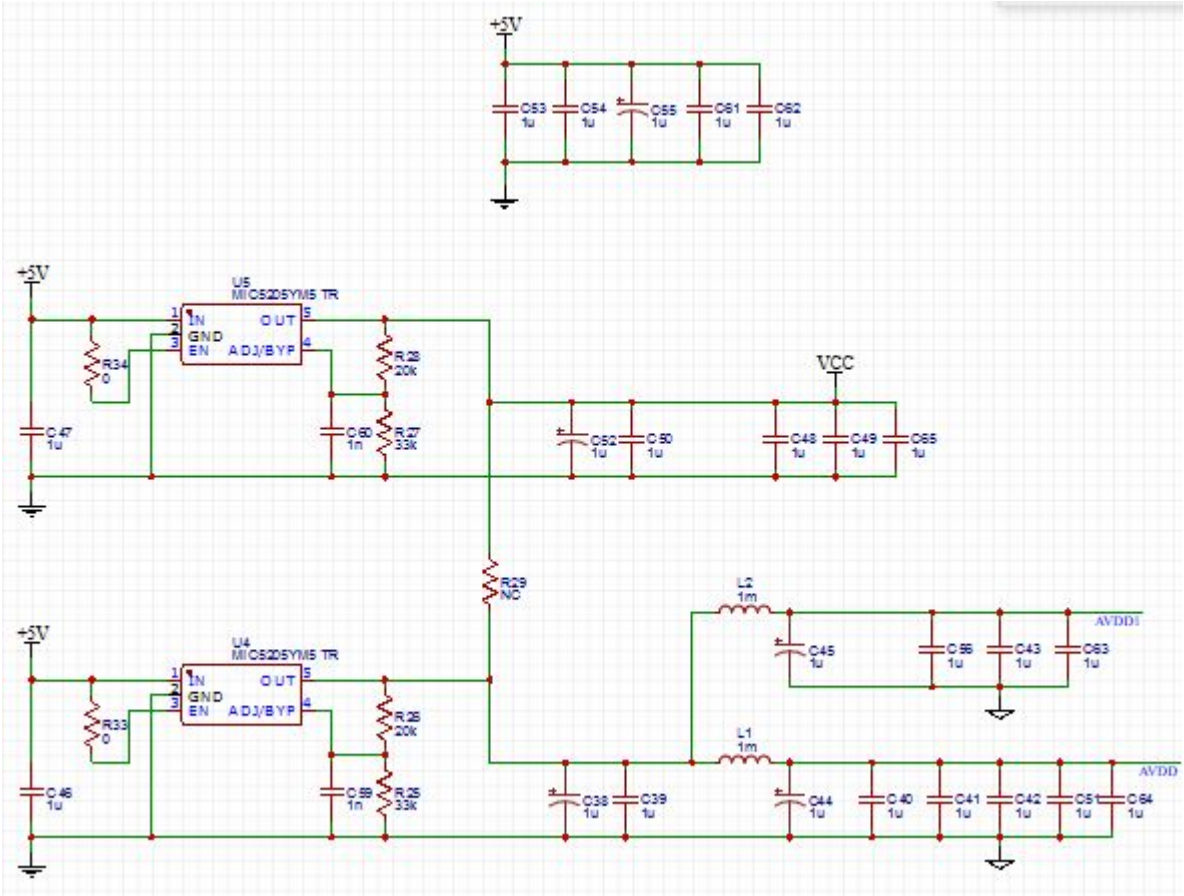

Figura 16 – Esquemático da fonte – Fonte: Autor.

Após projetar a parte dos filtros e a fonte de alimentação, o próximo passo foi o projeto do esquemático da placa principal. O esquemático elaborado pode ser observado na figura 17.

Para esta placa foi considerado criar uma interface entre os pinos de saída do ADS1298 e os pinos de entrada do microcomputador *Raspberry PI 3B*, cujas configurações foram especificadas em 3.5.1. Este processador foi escolhido devido ao poder de processamento ser mais do que o necessário para a realização da tarefa, possuir inúmeras opções de conexões externas e também pela estabilidade no mercado, pois, por mais que a linha *Raspberry* continue evoluindo, a fabricante sempre mantém os pinos de GPIO no mesmo formato, desta forma, a expansão futura do projeto será facilitada.

Desta forma, tanto os pinos de GPIO necessários quanto as portas do protocolo SPI do *chipset* do ADS1298 foram ligadas com seus respectivos pinos na placa do microcomputador.

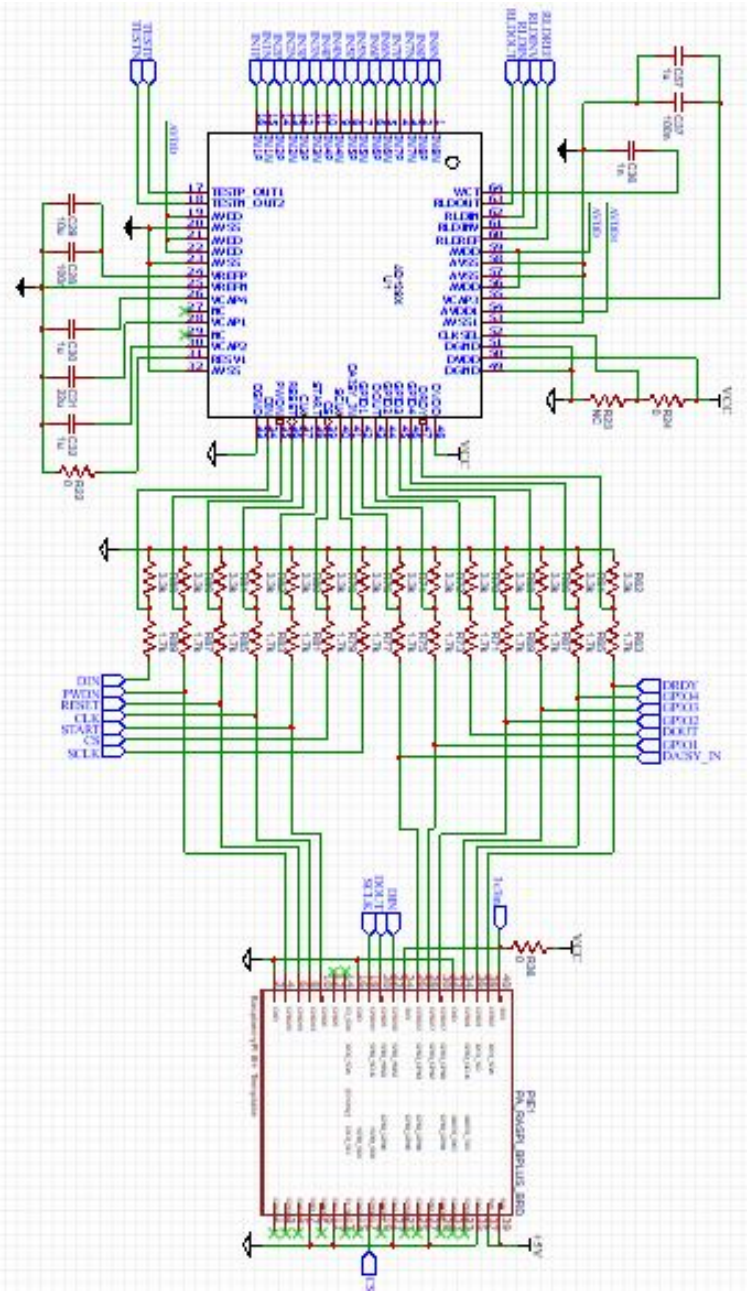

Figura 17 – Esquemático principal da placa – Fonte: Autor.

4.1.2 *Layout* da placa

Na figura 18 pode ser encontrado o *layout* final que foi produzido para a placa de aquisições de sinal. Em vermelho se encontra o plano terra da mesma, utilizado para reduzir a possibilidade de interferência de sinais externos a placa.

Na placa foi incluido uma saída USB, para o caso de alguma incompatibilidade com o *hardware* selecionado para o processamento ou mesmo possibilitar uma expansão alternativa para o projeto.

Isto foi feito utilizando o CI ATMEGA 32U4, o mesmo funciona como um conversor com entradas de SPI, convertendo-as para o formato USB padrão de computadores em geral,

permitindo uma expansão futura neste sentido. No momento, o setor para o mesmo já foi deixado na placa, porém, devido a incompatibilidade de tensões entre o mesmo e o *Raspberry PI*, foi mantido apenas o uso do microcomputador para esta fase do projeto.

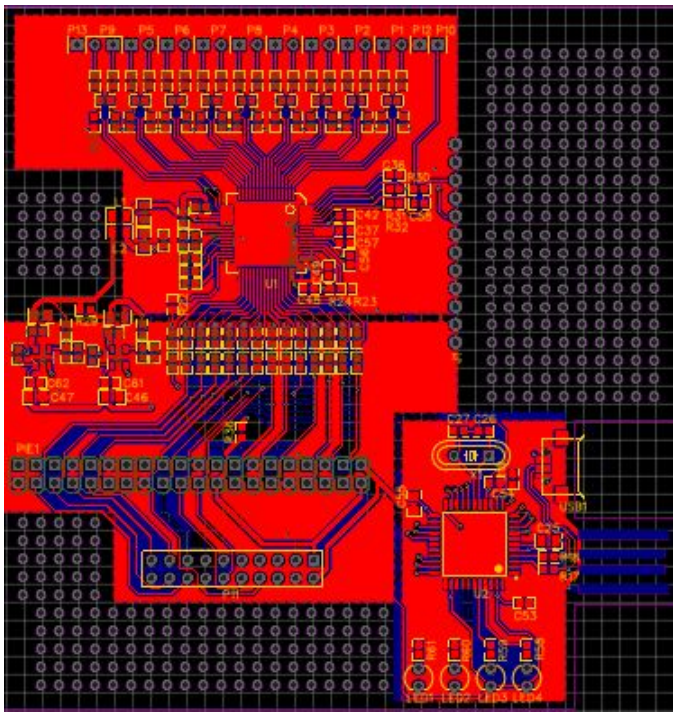

Figura 18 – Layout final da placa – Fonte: Autor.

Na tabela 1 é possível observar a lista de componentes que serão necessários para efetuar a montagem de duas das placas projetadas.

Tabela 1 – Componentes utilizados para confecção das placas.

Componente	Quantidade(und.)
Capacitor 01nF	02
Capacitor 10nF	36
Capacitor 20nF	18
Capacitor 100nF	04
Capacitor 01uF	60
Capacitor 10uF	02
Capacitor 22uF	02
Resistor 1kΩ	02
Resistor 1,7kΩ	28
Resistor 3,3kΩ	28
Resistor 16,9kΩ	02
Resistor 20,0kΩ	02
Resistor 33,3kΩ	04
Resistor 100,0kΩ	36
Indutor 1mH	04

Na tabela 2 pode-se observar o preço estimado para cada uma das partes que serão utilizadas para este trabalho. Os valores inseridos na planilha são do momento da aquisição de cada um dos componentes.

Tabela 2 – Componentes cotados - Sensores, Atuadores e Microcontroladores

Descrição	Preço
Eletrodos banhados a ouro	R$ 272,00
Pasta para EEG	R$ 44,90
Componentes SMD	R$ 171,20
CI ATMEGA32U4	R$ 14,75
CI ADS1298IPAG	R$ 344,90
Confecção da placa	R$ 52,80
Raspberry Pi 3 Modelo B	R$ 149,90
Total	R$ 1050,75

4.2 *SOFTWARE*

Encerrando a parte de *hardware* do processo de captura do sinal, a próxima área desenvolvida foi a parte de comunicação da placa de obtenção dos sinais com algum meio externo de processamento de dados. Para que esta parte possa ser configurada corretamente, primeiramente foi necessário realizar uma consulta ao *datasheet* do ADS1298 para saber quais comandos deveriam ser repassados ao mesmo para que fosse possivel trabalhar com o mesmo da melhor forma possível.

4.2.1 ADS1298 - Comandos

Figure 74. ID Control Register

7	6	5	4	3	2	1	0
DEV_ID[7:5]			1	0	DEV_ID[2:0]		
R-x			R-2h		R-x		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 17. ID Control Register Field Descriptions

Bit	Field	Type	Reset	Description
7:5	DEV_ID[7:5]	R	xh	**Device ID** These bits indicate the device family. 000 = Reserved 011 = Reserved 100 = ADS129x device family 101 = Reserved 110 = ADS129xR device family 111 = Reserved
4:3	RESERVED	R	2h	**Reserved** Always read back 2h
2:0	DEV_ID[2:0]	R	xh	**Channel ID** These bits indicates number of channels. 000 = 4-channel ADS1294 or ADS1294R 001 = 6-channel ADS1296 or ADS1296R 010 = 8-channel ADS1298 or ADS1298R 011 = Reserved 111 = Reserved

Figura 19 – Códigos de identificação de CIs – Fonte: (TEXAS INSTRUMENTS, 2019b).

O primeiro parâmetro observado é o identificador que o ADS1298 deve iniciar transmitindo. Este parâmetro serve para que seja possível confirmar durante o tempo de execução do código que o *slave* do arduíno é o ADS1298 e que desta forma o mesmo pode operar conforme planejado.

Conforme observado na figura 19, o identificador que o ADS1298 deve emitir seria o binário "1001 0010", que equivale ao hexadecimal "0x092", que é checado nas linhas a seguir do código do arduíno:

```
if(readReg(ID)!=0x92){
    while(1){
      delay(2000);
      Serial.println("Invalid device ID");
    }
  }
```

Caso o identificador não seja reconhecido, o código irá ter como *output* apenas a frase: *Invalid devide ID*.

Confirmando que é mesmo o ADS1298 que está conectado ao arduino, o próximo passo é configurar os três registradores internos do ADS1298 e também os registradores de cada canal individualmente.

Figure 75. CONFIG1: Configuration Register 1

7	6	5	4	3	2	1	0
HR	$\overline{\text{DAISY\_EN}}$	CLK_EN	0	0	DR[2:0]		
R/W-0h	R/W-0h	R/W-0h	R/W-0h		R/W-6h		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 18. Configuration Register 1 Field Descriptions

Bit	Field	Type	Reset	Description
7	HR	R/W	0h	**High-resolution or low-power mode** This bit determines whether the device runs in low-power or high-resolution mode. 0 = LP mode 1 = HR mode
6	$\overline{\text{DAISY\_EN}}$	R/W	0h	**Daisy-chain or multiple readback mode** This bit determines which mode is enabled. 0 = Daisy-chain mode 1 = Multiple readback mode
5	CLK_EN	R/W	0h	**CLK connection**(1) This bit determines if the internal oscillator signal is connected to the CLK pin when the CLKSEL pin = 1. 0 = Oscillator clock output disabled 1 = Oscillator clock output enabled
4:3	RESERVED	R/W	0h	**Reserved** Always write 0h
2:0	DR[2:0]	R/W	6h	**Output data rate** For High-Resolution mode, $f_{MOD} = f_{CLK} / 4$. For low power mode, $f_{MOD} = f_{CLK} / 8$. These bits determine the output data rate of the device. 000: $f_{MOD} / 16$ (HR Mode: 32 kSPS, LP Mode: 16 kSPS) 001: $f_{MOD} / 32$ (HR Mode: 16 kSPS, LP Mode: 8 kSPS) 010: $f_{MOD} / 64$ (HR Mode: 8 kSPS, LP Mode: 4 kSPS) 011: $f_{MOD} / 128$ (HR Mode: 4 kSPS, LP Mode: 2 kSPS) 100: $f_{MOD} / 256$ (HR Mode: 2 kSPS, LP Mode: 1 kSPS) 101: $f_{MOD} / 512$ (HR Mode: 1 kSPS, LP Mode: 500 SPS) 110: $f_{MOD} / 1024$ (HR Mode: 500 SPS, LP Mode: 250 SPS) 111: Reserved (do not use)

Figura 20 – Comandos do primeiro registrador – Fonte: (TEXAS INSTRUMENTS, 2019b).

Na figura 20 podem ser observadas todas as possiveis configurações para o primeiro registrador do ADS1298. Neste registrador deve-se determinar se o dispositivo irá atuar em modo de alta precisão ou em modo econômico. Ainda neste registrador deve-se determinar o modo de operação do *clock*, do parâmetro de *daisy* e o *data rate output* do CI. Nas linhas abaixo pode-se observar o código que configura este registrador.

```
writeReg(CONFIG1,
    (1<<7)|      //Modo de alta resolução
    (6)|         //fMod/1024 -> 500SPS
    0);          //Daisy chain, oscilador externo desabilitado
```

Na figura 21 podem ser observadas as configurações do segundo registrador do ADS1298. Este registrador serve para configurar todos os parâmetros do teste interno que o ADS1298 é capaz de rodar.

Figure 76. CONFIG2: Configuration Register 2

7	6	5	4	3	2	1	0
0	0	WCT_CHOP	INT_TEST	0	TEST_AMP	TEST_FREQ[1:0]	
R/W-1h		R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 19. Configuration Register 2 Field Descriptions

Bit	Field	Type	Reset	Description
7:6	RESERVED	R/W	1h	**Reserved** Always write 0h
5	WCT_CHOP	R/W	0h	**WCT chopping scheme** This bit determines whether the chopping frequency of WCT amplifiers is variable or fixed. 0 = Chopping frequency varies, see Table 7 1 = Chopping frequency constant at f_{MOD} / 16
4	INT_TEST	R/W	0h	**TEST source** This bit determines the source for the test signal. 0 = Test signals are driven externally 1 = Test signals are generated internally
3	RESERVED	R/W	0h	**Reserved** Always write 0h
2	TEST_AMP	R/W	0h	**Test signal amplitude** These bits determine the calibration signal amplitude. 0 = 1 × –(VREFP – VREFN) / 2400 V 1 = 2 × –(VREFP – VREFN) / 2400 V
1:0	TEST_FREQ[1:0]	R/W	0h	**Test signal frequency** These bits determine the calibration signal frequency. 00 = Pulsed at f_{CLK} / 2^{21} 01 = Pulsed at f_{CLK} / 2^{20} 10 = Not used 11 = At dc

Figura 21 – Comandos do segundo registrador – Fonte: (TEXAS INSTRUMENTS, 2019b).

Nas linhas de código abaixo está como foi deixado a configuração do segundo registrador para este projeto. Este teste será rodado posteriormente e apresentado em 5.

```
writeReg(CONFIG2,
    (1<<4)|      //Sinais de testes internos
    (0)|         //Pulsado em fClk/2^21
    0);          //Chopping at Fmod/128=4kHz, testAmp=1*VRef/2400
```

O terceiro registrador por sua vez serve para configurar as *features* extras do ADS1298, como o *buffer* interno de referência, a tensão de referência para as medidas, e as *features* referentes ao RLD.

Figure 77. CONFIG3: Configuration Register 3

7	6	5	4	3	2	1	0
$\overline{\text{PD\_REFBUF}}$	1	VREF_4V	RLD_MEAS	RLDREF_INT	$\overline{\text{PD\_RLD}}$	RLD_LOFF_SENS	RLD_STAT
R/W-0h	R/W-1h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 20. Configuration Register 3 Field Descriptions

Bit	Field	Type	Reset	Description
7	$\overline{\text{PD\_REFBUF}}$	R/W	0h	**Power-down reference buffer** This bit determines the power-down reference buffer state. 0 = Power-down internal reference buffer 1 = Enable internal reference buffer
6	RESERVED	R/W	1h	**Reserved** Always write 1h
5	VREF_4V	R/W	0h	**Reference voltage** This bit determines the reference voltage, VREFP. 0 = VREFP is set to 2.4 V 1 = VREFP is set to 4 V (use only with a 5-V analog supply)
4	RLD_MEAS	R/W	0h	**RLD measurement** This bit enables RLD measurement. The RLD signal may be measured with any channel. 0 = Open 1 = RLD_IN signal is routed to the channel that has the MUX_Setting *010* (V_{REF})
3	RLDREF_INT	R/W	0h	**RLDREF signal** This bit determines the RLDREF signal source. 0 = RLDREF signal fed externally 1 = RLDREF signal (AVDD – AVSS) / 2 generated internally
2	$\overline{\text{PD\_RLD}}$	R/W	0h	**RLD buffer power** This bit determines the RLD buffer power state. 0 = RLD buffer is powered down 1 = RLD buffer is enabled
1	RLD_LOFF_SENS	R/W	0h	**RLD sense function** This bit enables the RLD sense function. 0 = RLD sense is disabled 1 = RLD sense is enabled
0	RLD_STAT	R	0h	**RLD lead-off status** This bit determines the RLD status. 0 = RLD is connected 1 = RLD is not connected

Figura 22 – Comandos do terceiro registrador – Fonte: (TEXAS INSTRUMENTS, 2019b).

Nas linhas abaixo está exposto como ficaram as configurações para este registrador.

```
writeReg(CONFIG3,
  (1<<7)|      //Buffer de referencia interna ativo
  (1<<6)|      //Sempre 1
  0);          //VRef em 2.4V, RLD aberto, RLDREF externo, RLD buffer desligado,
               //RLD desconectado.
```

Por ultimo, na figura 23 estão expostas as configurações que cada um dos canais podem receber. Neste registrador é possivel ativar ou desativar algum canal específico, configurar o ganho individual de cada canal e também configurar a qual tipo de eletrodo ou dispositivo este canal foi conectado. Vale lembrar que cada canal possuí o seu registrador, logo, todos devem ser corretamente configurados para que o ADS1298 funcione como esperado.

Figure 79. CHnSET: Individual Channel Settings Register

7	6	5	4	3	2	1	0
PD*n*	GAIN*n*[2:0]			0	MUX*n*[2:0]		
R/W-0h	R/W-0h			R/W-0h	R/W-0h		

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Table 22. Individual Channel Settings (*n* = 1 to 8) Field Descriptions

Bit	Field	Type	Reset	Description
7	PD*n*	R/W	0h	**Power-down** This bit determines the channel power mode for the corresponding channel. 0 = Normal operation 1 = Channel power-down. When powering down a channel, TI recommends that the channel be set to input short by setting the appropriate MUXn[2:0] = 001 of the CHnSET register.
6:4	GAIN*n*[2:0]	R/W	0h	**PGA gain** These bits determine the PGA gain setting. 000 = 6 001 = 1 010 = 2 011 = 3 100 = 4 101 = 8 110 = 12
3	RESERVED	R/W	0h	**Reserved** Always write 0h
2:0	MUX*n*[2:0]	R/W	0h	**Channel input** These bits determine the channel input selection. 000 = Normal electrode input 001 = Input shorted (for offset or noise measurements) 010 = Used in conjunction with RLD_MEAS bit for RLD measurements. See the *Right Leg Drive (RLD) DC Bias Circuit* subsection of the *ECG-Specific Functions* section for more details. 011 = MVDD for supply measurement 100 = Temperature sensor 101 = Test signal 110 = RLD_DRP (positive electrode is the driver) 111 = RLD_DRN (negative electrode is the driver)

Figura 23 – Comandos do registrador do canal – Fonte: (TEXAS INSTRUMENTS, 2019b).

Nas linhas abaixo é possível observar como os canais foram configurados para este projeto.

```
  writeReg(CH1SET,
^^I(1<<4)|      //Ganho do canal = 1
    0);          //Canal ligado,  eletrodo normal
  writeReg(CH2SET, (1<<4)| 0);
  writeReg(CH3SET, (1<<4)| 0);
  writeReg(CH4SET, (1<<4)| 0);
  writeReg(CH5SET, (1<<4)| 0);
  writeReg(CH6SET, (1<<4)| 0);
  writeReg(CH7SET, (1<<4)| 0);
  writeReg(CH8SET, (1<<4)| 0);
```

4.2.2 SPI

Para esta monografia, foi seguido duas linhas de compatibilidade com placas para realizarem o processamento externo:

- *Raspberry Pi*;
- Computador externo através da interface do arduino;

Para a parte da comunicação com o computador através da interface do arduíno, foi utilizado a biblioteca SPI nativa do arduino para realizar a configuração do protocolo e transmitir os dados.

A configuração da mesma é realizada através das seguintes linhas de código:

```
void SPI_startup(){
  SPI.begin();
  SPI.beginTransaction(SPISettings(1945000, MSBFIRST, SPI_MODE1));
}
```

Na primeira linha um elemento da classe SPI é iniciado e na segunda seus parâmetros são setados para um *clock* de 1.945MHz, este valor foi escolhido por ser suficiente para realizar todas as transações necessárias e ainda estar dentro da capacidade de ambos os CIs trabalharem. O segundo parâmetro se refere ao modo como serão transmitidos os dados entre os CIs, que, para este caso, será iniciando a transmissão pelo *Most Significant Bit* (MSB) do dado que será transmitido, este modo de transmissão está especificado no *datasheet* do ADS1298. O terceiro parâmetro se refere a polaridade do *clock*, que especifica quando irá ocorrer a captura do dado, que neste caso, será na borda de descida do *clock*.

As linhas a seguir definem as operações de transação que serão realizadas pelo Atmega através da SPI. Os dois primeiros comandos, SPI_START e SPI_STOP operam em cima do pino CS do ADS1298. Este pino serve para indicar para o ADS quando o mesmo deve esperar, receber ou emitir um sinal. Sempre que será realizada uma transição de dados entre *slave* e *master* este pino deve ser colocado em nível lógico baixo.

Os dois últimos comandos servem para transferir dados entre o Atmega e o ADS1298 através do protocolo SPI.

```
#define SPI_START()          digitalWrite(CS_PIN,LOW)
#define SPI_STOP()           digitalWrite(CS_PIN,HIGH)
#define SPI_TRANSFER1(b)     SPI.transfer(b)
#define SPI_TRANSFERN(a,s)   SPI.transfer(a,s)
```

Porém, mesmo sendo simples de usá-los separadamente, ainda deve-se respeitar uma sequência específica de operações para passar comandos entre as placas e para que a troca ocorra adequadamente. Pensando nisto, foram implementadas as funções *readReg*, *writeReg* e *command*. Estas funções serão explicadas na seção 4.2.3.

Já para a comunicação através da placa *raspberry* pi foi utilizada a biblioteca spidev, que fornece uma implementação muito parecida com a implementação descrita anteriormente para o arduino, porém de uma forma mais simples.

Nas linhas a seguir se encontra o código necessário para a correta inicialização da SPI da Raspberry Pi. Vale ter em mente que a Raspberry Pi pode atuar apenas como *master* de uma comunicação SPI. Inicialmente deve-se criar um objeto da classe SPI, configura-lo conforme o *slave* foi ligado com o comando open(0,X), sendo X 0 ou 1, dependendo da porta onde o *slave* está conectado.

A configuração continua repassando a velocidade máxima a qual o barramento do SPI irá operar, para este caso foi escolhido a mesma do arduíno, o modo como a SPI irá funcionar, que deverá ser o mesmo do arduíno, porém aqui deve-se especificar qual a polaridade e o *alpha* do *clock*, conforme feito abaixo. Por último, deve-se fixar o valor do parâmetro *cshigh* como alto, para que a *Raspberry* cuide deste pino para o programador.

```
def SPI_startup():
  spi = spidev.SpiDev()
  spi.open(0,0)
  spi.max_speed_hz = 1945000
  spi.mode = 0b01    #[CPOL|CPHA],
  spi.cshigh = False
  return spi
```

Para esta implementação não é necessário se preocupar com o pino CS pois o comando *xfer* desta biblioteca já efetua o controle do mesmo corretamente, logo, as quatro funções que eram necessárias no arduino podem ser reduzidas a apenas uma linha de comando dentro do python, conforme visto a seguir.

```
  ret = spi.xfer([RREG | reg,0,0])
```

Durante os testes realizados com a placa 3B da Raspberry Pi, acabou-se por descobrir que, mesmo que fossem adicionadas interrupções via *hardware* para os pinos que disparavam a leitura dos dados, o sistema operacional Raspbean não é capaz de operar com o fluxo de *inputs* e *outputs* deste projeto.

Esta limitação causa problemas ao salvar valores para os registradores do ADS1298, o que impossibilita a total e correta configuração do mesmo, além disto, alguns dos *bits* durante a leitura dos dados eram trocados de posição. Estas limitações acabam tornando a Raspberry Pi uma opção inviável para o desenvolvimento do projeto dentro deste sistema operacional. Não foram testados outros sistemas operacionais, porém um sistema que trabalhe em tempo-real pode ser uma opção para solucionar este problema.

A fim de tornar algum possivel futuro trabalho mais simples, o código desenvolvido para a Raspberry Pi foi deixado na integra nas sessões B.0.5, B.0.6 e B.0.7. Lá será encontrado um código para trabalhar com pinos de GPIO da Raspberry Pi, lendo o SPI, convertendo e trabalhando com ela para gerar os gráficos desenvolvidos neste projeto na sessão 4.2.4.3.

4.2.3 Código Arduino

Após configurar corretamente o modo operante do protocolo SPI os comandos descritos na seção 4.2.1 foram implementados num arquivo específico para eles. O código deste arquivo se encontra em B.0.4.

Neste arquivo se encontram os endereços dos registradores internos ao ADS1298 e os comandos do mesmo mapeados em variáveis do tipo *enumerates* junto com a definição de três funções primordiais para o uso do SPI e uma para a leitura dos dados. São elas:

- readReg;
- writeReg;
- command;
- readConversation.

A primeira função, *readReg*, implementa todos os passos necessários para efetuar a leitura do valor que foi salvo em um dos registradores internos do ADS1298, para isto, basta passar o endereço do registrador através do parâmetro "reg"da mesma. Ela utiliza como base as funções implementadas para o uso do SPI, descritas em 4.2.2.

```
uint8_t readReg(Registers reg){
    uint8_t ret;
    SPI_START();
    SPI_TRANSFER1(RREG | reg);
    SPI_TRANSFER1(0);
    ret = SPI_TRANSFER1(0);
    SPI_STOP();
    return ret;
}
```

Já a segunda função possuí a funcionalidade de escrever em um dos registradores apontados pelo usuário através do parâmetro "reg"o valor desejado para o mesmo, que deve ser passado através do parâmetro "val". Esta função também utiliza como base as funções descritas em 4.2.2.

```
void writeReg(Registers reg, uint8_t val){
    uint8_t ret;
    SPI_START();
    SPI_TRANSFER1(WREG | reg);
    SPI_TRANSFER1(0);
    SPI_TRANSFER1(val);
    SPI_STOP();
```

```
    return ret;
}
```

A terceira função serve para repassar comandos desejados para o ADS1298 através do parâmetro de entrada "c".

```
void command(Commands c){
    SPI_START();
    SPI_TRANSFER1(c);
    SPI_STOP();
}
```

Por último, ainda dentro deste arquivo, se encontra a função *readConversion()*. Esta função é responsável por receber os 24 *bits* do ADS1298, montar um número a partir deles e salvar seu valor dentro de uma casa do vetor de leituras *eegLastReading*.

```
int32_t eegLastReading[9];
void readConversion(){
    SPI_START();
    SPI_TRANSFER1(RDATA);
    for(int i=0; i<9; i++){
      eegLastReading[i] = ((uint32_t)SPI_TRANSFER1(0))<<16;
      eegLastReading[i] |= ((uint32_t)SPI_TRANSFER1(0))<<8;
      eegLastReading[i] |= ((uint32_t)SPI_TRANSFER1(0))<<0;
      if(eegLastReading[i] & (1UL<<23))eegLastReading[i]|=0xFF000000UL;
    }
    SPI_STOP();
}
```

Um segundo arquivo montado com o código do arduíno, cujo código se encontra em B.0.3, é responsável por armazenar quais pinos estão ligados as mais diferentes portas utilizadas pelo código.

Neste arquivo também estão descritas duas funções, são elas:

- GPIO_startup;
- ADS_hardReset;

A primeira função tem o papel de inicializar todos os GPIO's que serão utilizados neste código. Nas linhas abaixo pode-se ver o código implementado nela.

```
void GPIO_startup(){
  pinMode(PWDN_PIN,     OUTPUT);
  pinMode(RESET_PIN,    OUTPUT);
```

```
  pinMode(CLK_PIN,      INPUT);
  pinMode(START_PIN,    OUTPUT);
  pinMode(DAISY_IN_PIN, OUTPUT);
  pinMode(DRDY_PIN,     INPUT);
  pinMode(CS_PIN,       OUTPUT);
  pinMode(AD_GPIO1_PIN, INPUT);
  pinMode(AD_GPIO2_PIN, INPUT);
  pinMode(AD_GPIO3_PIN, INPUT);
  pinMode(AD_GPIO4_PIN, INPUT);
  pinMode(LED0_PIN,     OUTPUT);
  pinMode(LED1_PIN,     OUTPUT);
  digitalWrite(PWDN_PIN,     HIGH);
  digitalWrite(RESET_PIN,    HIGH);
  digitalWrite(START_PIN,    LOW);
  digitalWrite(DAISY_IN_PIN, LOW);
  digitalWrite(CS_PIN,       HIGH);
  digitalWrite(AD_GPIO1_PIN, LOW);
  digitalWrite(AD_GPIO2_PIN, LOW);
  digitalWrite(AD_GPIO3_PIN, LOW);
  digitalWrite(AD_GPIO4_PIN, LOW);
  digitalWrite(LED0_PIN,     LOW);
  digitalWrite(LED1_PIN,     LOW);
}
```

E a última função definida neste arquivo seria a função *ADS_hardReset*. Esta função é responsável por realizar um *reset* de *hardware* no CI do ADS1298. Isto garante que quando a placa seja ligada, o estado do ADS seja reiniciado. O código responsável por isto se encontra nas linhas abaixo.

```
void ADS_hardReset(){
  //Hard Reset
  delay(150);
  digitalWrite(RESET_PIN, LOW);
  delay(1);
  digitalWrite(RESET_PIN, HIGH);
  delay(1);
}
```

Por ultimo, existe a função principal, exposta em B.0.2. Esta função é responsável por chamar todas as funções que foram explicadas anteriormente, inicializando corretamente

todas as GPIO's da mesma, inicializando a interface SPI, realizando a confirmação de que é o *slave* correto, configurando todos os registradores do ADS1298 e, por último, ela se prende em um laço responsável por efetuar a leitura dos dados de *output* do ADS1298 e comunica-los através da porta serial do arduino. O código deste laço principal está exposto nas linhas abaixo.

```
void loop() {
  while(digitalRead(DRDY_PIN));
  digitalWrite(LED0_PIN,     HIGH);
  readConversion();
  int i = 0;
  for(i = 1; i<9;i++){
    Serial.print(eegLastReading[i], HEX);
^^ISerial.print(',',HEX);
  }
  Serial.println();
  digitalWrite(LED0_PIN,     LOW);
```

4.2.4 Python

Python é uma ferramenta muito poderosa e capaz de simplificar alguns processos cruciais, sem perder alguns parâmetros de otimização, desde que esteja bem implementado. Pensando neste cenário, esta linguagem foi escolhida para construir a parte que irá rodar em cima de um computador.

Como para esta atividade era necessário que o programa em Python se comportasse da maneira mais ágil possível, algumas estratégias foram utilizadas a fim de otimizar diversos processos para que fosse possível extrair o melhor desempenho possível.

A primeira estratégia cogitada pelo autor foi utilizar o conceito de *multithreading* para construir uma estrutura consumidor-produtor a fim de evitar perda de dados na leitura dos mesmos. Porém, como se tratava de python, isto não era possível, uma vez que o interpretador de códigos do python atrela sua execução a um *process_id* e, por mais que permita emular *multi-threads*, estas acabam tendo seu processamento linearmente, então elas não são capazes de criar um paralelismo durante a execução.

Tendo esta dificuldade, a solução encontrada foi de criar o paralelismo através de *multiprocessing*, onde o código subiria alguns processos diferentes para os diferentes trechos de código e, desta forma, é possível gerar processamento em paralelo através do python.

Para executar esta atividade foi utilizada a biblioteca *multiprocessing* do python, que permitiu a criação destes processos.

A arquitetura final do programa utiliza três processos para construir o paralelismo, são eles:

- productor;

- graf;
- consumer;

O código total destes processos se encontra em B.0.1 e nas próximas sessões serão descritos o funcionamento individual de cada um destes processos. O código para iniciar estes três processos e sincronizar com um deles está nas linhas a seguir. Foi utilizado o processo gráfico como processo para sincronização final, através do comando *join* para que, quando o usuário fechasse a janela grafica, o programa se encerrasse.

```
if __name__ == '__main__':
    with Manager() as manager:
        d = manager.dict()
        history = manager.list()
        history_graf = manager.list()
        prod = Process(target=productor, args=(history,))
        prod.start()
        cons = Process(target=consumer, args=(history,history_graf))
        cons.start()
        cons_graf = Process(target=graf, args=(history_graf,))
        cons_graf.start()
        cons_graf.join()
```

4.2.4.1 USB

O primeiro processo é responsável pela leitura da porta USB em que a placa está conectada. Foi convencionado que a placa sempre seria conectada na mesma placa no computador, então a mesma está definida de forma fixa, porém a mesma pode ser mudada.

Para efetuar a leitura corretamente é utilizada a biblioteca "serial"em conjunto com a biblioteca "io"do python.

O código para esta parte pode ser encontrado nas linhas a seguir. Nele, é inicializado um objeto do tipo serial, com os parâmetros de porta USB, velocidade da comunicação e tempo limite para leitura, em segundos. Após a inicialização desta serial, os dados são lidos, transformados para o formato decimal e então anexados em uma fila que será lida pelo processo responsável pela implementação do filtro digital e do *backup*.

```
def productor(fila):
    eegReading = [0]*8
    ser = serial.Serial('COM4',1000000,timeout=1)
    if(ser.is_open):
        str(ser.readline())
        while(True):
```

```
            line = str(ser.readline()).replace('b\'','')
            lineRead = line.split(',')
            for i in range(8):
                lineRead[i] = int(lineRead[i],16)
            for i in range(8):
                if lineRead[i] > 0x800000:
                    lineRead[i] -= 0x1000000
                eegReading[i] = lineRead[i]*3.3/0x800000
            fila.append(eegReading)
    ser.close()
```

4.2.4.2 Implementação do filtro e dados de *backup*

O segundo processo, *consumer*, é responsável por retirar os dados da primeira fila, separa-los por canal, aplicar um filtro FIR de janela de Blackman individualmente em cada canal, salvar estes dados em um arquivo no formato *comma separated values* (CSV) e por último coloca-los em uma segunda fila, que será consultada pelo terceiro processo, responsável por montar a parte gráfica do projeto, onde as ondas serão exibidas.

Para montar o filtro FIR com janelamento por Blackman foi utilizado a biblioteca signal, que participa do pacote de bibliotecas SciPy. Estas bibliotecas são otimizadas para alta performance, isto permite que sejam utilizados filtros de maiores ordens sem que o programa apresente perda de performance. Isto ocorre principalmente por uma boa parcela das bibliotecas desse pacote serem implementadas em C por parte do compilador, o que garante que o código tenha um ótimo rendimento.

O código necessário para montar o filtro está exposto nas linhas a seguir, nele, o filtro foi definido com sua frequência de corte em 40 Hz, frequência de amostragem de 500 SPS e de 50ª ordem.

```
ordem = 50
f = 40
fs = 500
black = signal.firwin((ordem+1), f, window='blackman', nyq = fs/2)
comp = sum(black)
```

E para aplicar o filtro no sinal:

```
            pontos_filtrados_ch1 = \
                signal.convolve(history_ch1, black, mode='same')/comp
            pontos_filtrados_ch2 = \
                signal.convolve(history_ch2, black, mode='same')/comp
            pontos_filtrados_ch3 = \
```

```
    signal.convolve(history_ch3, black, mode='same')/comp
pontos_filtrados_ch4 = \
    signal.convolve(history_ch4, black, mode='same')/comp
pontos_filtrados_ch5 = \
    signal.convolve(history_ch5, black, mode='same')/comp
pontos_filtrados_ch6 = \
    signal.convolve(history_ch6, black, mode='same')/comp
pontos_filtrados_ch7 = \
    signal.convolve(history_ch7, black, mode='same')/comp
pontos_filtrados_ch8 = \
    signal.convolve(history_ch8, black, mode='same')/comp
```

Esta técnica de filtro digital foi escolhida devido ao baixo espalhamento de dados que a mesma consegue com ordens menores em relação ao filtro FIR por Hanning e Blackman para os cenários aos quais elas foram expostas, aliado a taxa de rejeição elevada que este filtro tem para frequências fora de sua banda de passagem.

Isto foi concluído após realizar testes com o sinal captado pela placa. Estes testes podem ser verificados nas imagens 24, 25, 26, 27 e 28.

Pode-se notar que, com um filtro de baixa ordem o processamento digital não consegue ser efetivo suficiente para filtrar adequadamente o sinal com nenhuma das técnicas, como visto na figura 24, porém, ao se ampliar esta ordem pode-se notar uma grande melhoria no sinal, como observado na figura 28.

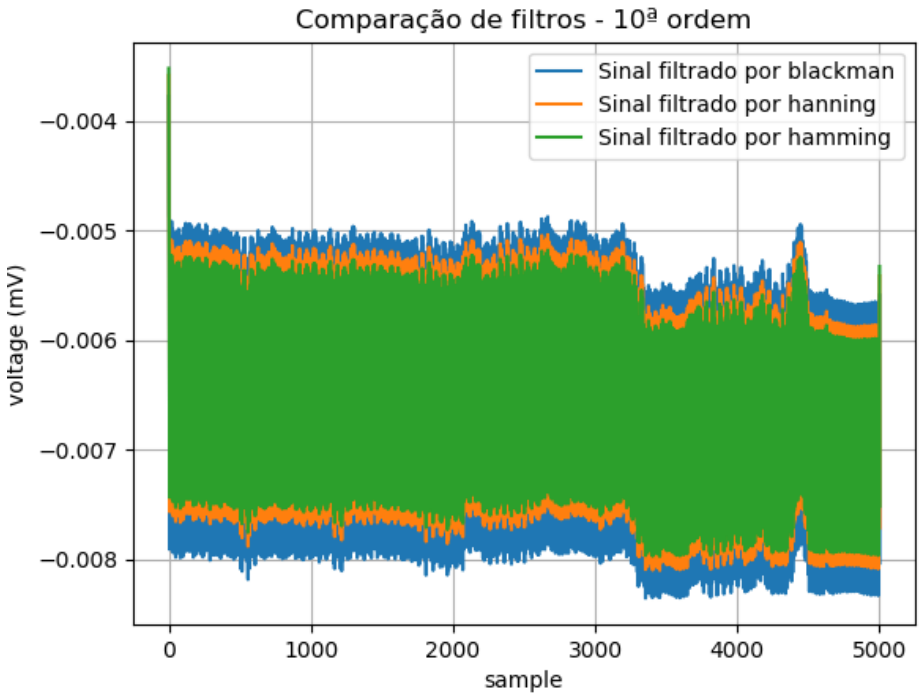

Figura 24 – Comparação do sinal filtrado com filtros de décima ordem – Fonte: Autor.

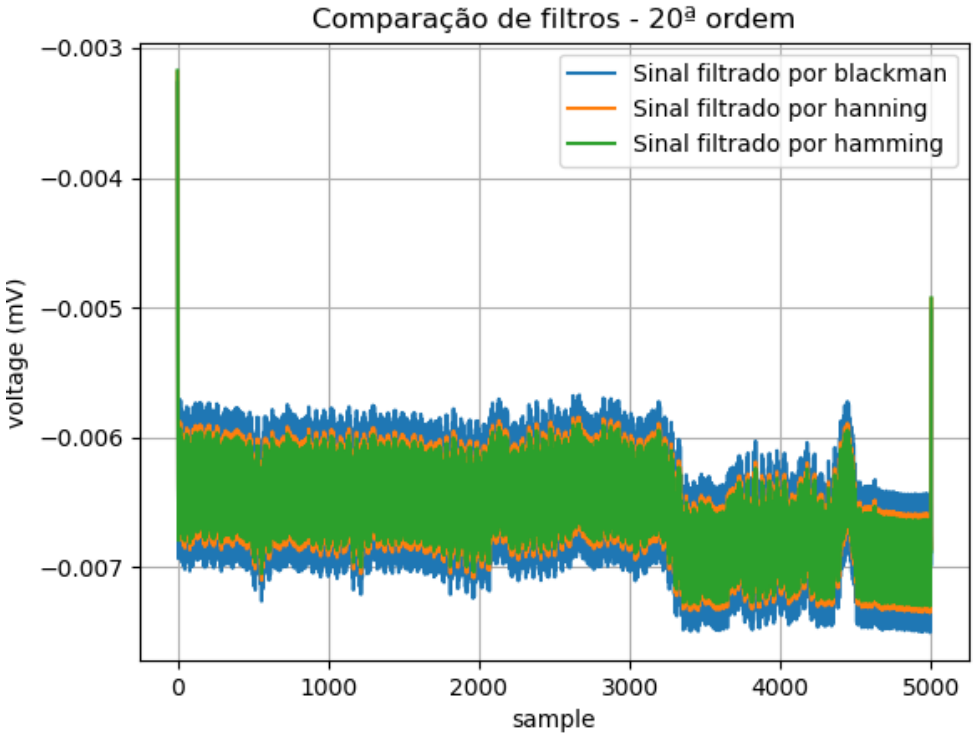

Figura 25 – Comparação do sinal filtrado com filtros de vigésima ordem – Fonte: Autor.

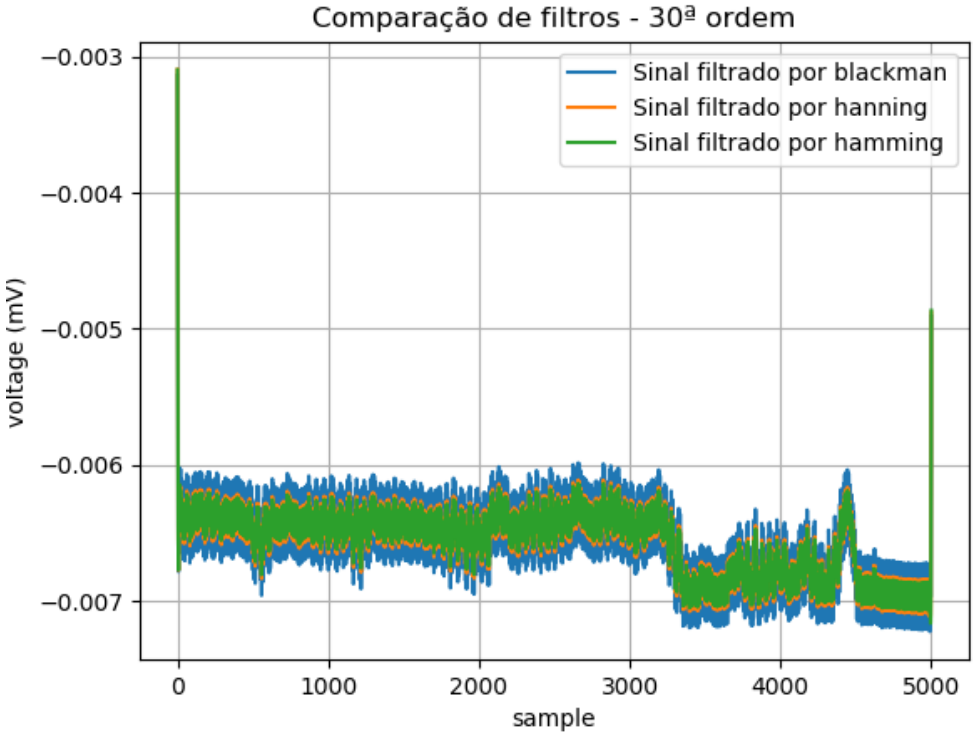

Figura 26 – Comparação do sinal filtrado com filtros de trigésima ordem – Fonte: Autor.

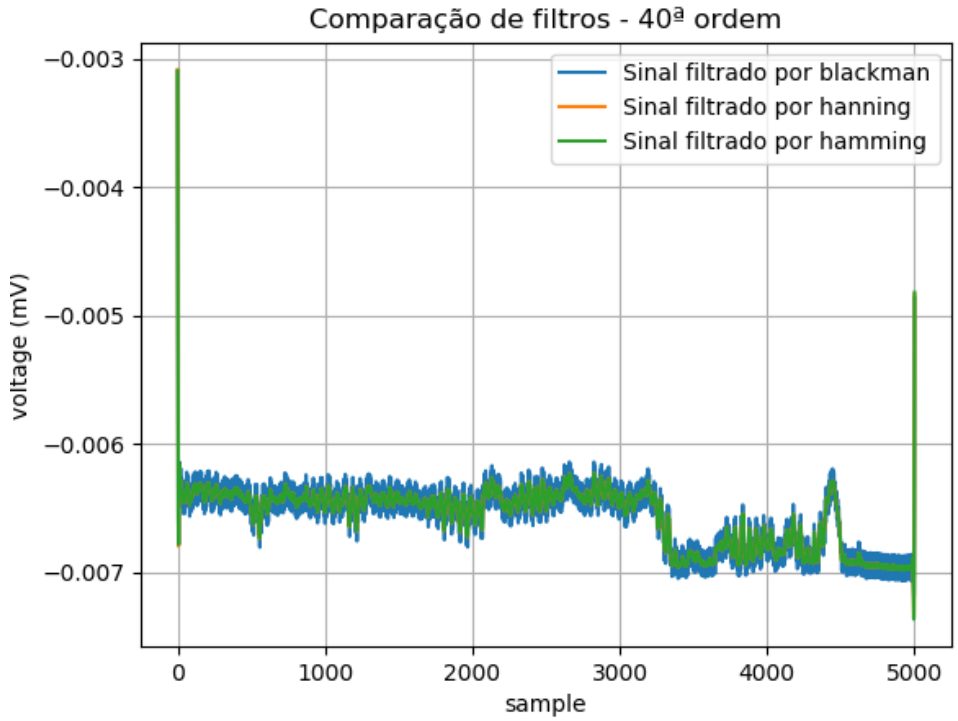

Figura 27 – Comparação do sinal filtrado com filtros de quadragésima ordem – Fonte: Autor.

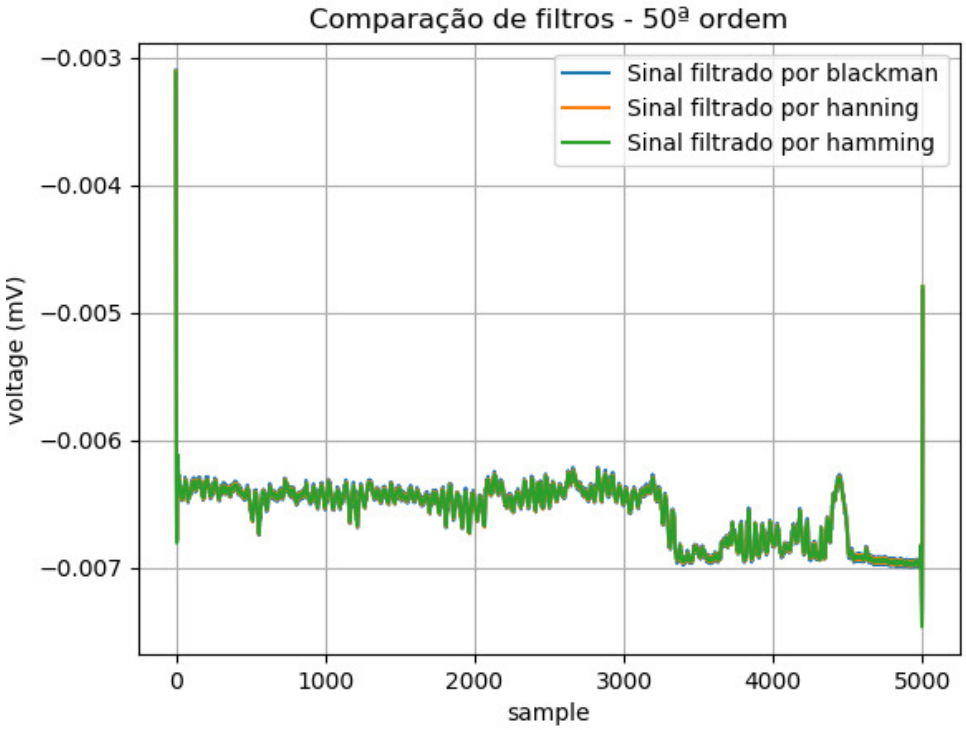

Figura 28 – Comparação do sinal filtrado com filtros de quinquagésima ordem – Fonte: Autor.

A segunda parte deste processo é retransmitir estes dados filtrados para o processo gráfico e salva-los em disco.

A parte responsável por esta atividade está exposta nas linhas abaixo. Para escrever no disco foi utilizado o leitor de arquivos padrão do python.

```
for x in range(len(pontos_filtrados_ch1)):
    linha_pos_filtro=[pontos_filtrados_ch1[x],\
        pontos_filtrados_ch2[x],pontos_filtrados_ch3[x],\
        pontos_filtrados_ch4[x],pontos_filtrados_ch5[x],\
        pontos_filtrados_ch6[x],pontos_filtrados_ch7[x],\
        pontos_filtrados_ch8[x]]
    fila_graf.append(linha_pos_filtro)
    arq = open('simulado.csv', 'a')
    arq.write(str(linha_pos_filtro).replace('[','')\
        .replace(']','').replace(' ','')+'\n')
    arq.close()
```

4.2.4.3 Parte gráfica

Para esta parte foi utilizada uma implementação da biblioteca gráfica *matplotlib* chamada *animation*. Esta parte da biblioteca permite gerar gráficos dinâmicos que se atualizam a cada intervalo de tempo, conforme especificado na hora de cria-lo pela primeira vez.

Pensando ainda em agilizar esta parte, foi utilizado o tipo de variavel *deque* que se trata de uma lista com tamanho fixo otimizada para realizar operações de inserção e remoção de itens dentro da mesma. Há também um ganho de velocidade na hora de consultar os valores contidos na mesma. Logo, a utilização deste tipo de variáveis para guardar os pontos que serão guardados nos gráficos é muito vantajosa em termos de agilidade de execução.

Quanto ao funcionamento da biblioteca *animated*, ela executa uma função definida pelo programador a cada X milisegundos, esta função deve atualizar os pontos do gráfico, que serão replotados após isto. Este gráfico deve ser modelado como um gráfico padrão do *matplotlib*. O gráfico responsável por modelar o gráfico é:

```
lim_inf_y = -0.02
lim_max_y = 0.02
p_max = 0.1+0.001
p_min = -0.1
p_passos = 0.1
window_size = 1500
grid_on = True
line_style_1 = 'g-'
line_style_2 = 'b-'
{...}
fig = plt.figure()
```

```
    axs = fig.subplots(8, 1, sharex=True)
    fig.subplots_adjust(hspace=0)
    fig.suptitle('Canais EEG', fontsize=30)

    plt_1, = axs[0].plot(channel_1, line_style_1, label='Channel 1')
    axs[0].set_ylim(lim_inf_y,lim_max_y)
    axs[0].set_ylabel('Channel 1')
    axs[0].grid(grid_on)
    axs[0].set_yticks(np.arange(p_min, p_max, p_passos))

    plt_2, = axs[1].plot(channel_2, line_style_2, label='Channel 2')
    axs[1].set_ylim(lim_inf_y,lim_max_y)
    axs[1].set_ylabel('Channel 2')
    axs[1].grid(grid_on)
    axs[1].set_yticks(np.arange(p_min, p_max, p_passos))
    {...}
```

Foi colocado como exemplo apenas a configuração de dois dos oito *subplots*, porém o código poderá ser visto na integra em B.0.1.

Para configurar o funcionamento da animação deve ser utilizado o seguinte código:

```
ani = animation.FuncAnimation(fig, animate, interval=10)
```

Onde deve-se passar como parâmetros a figura onde a função irá operar, a função de alteração dos *plots* e o intervalo em milisegundos da atualização do gráfico.

A função de animação para a atualização deste gráfico é:

```
def animate(i):
        for lec in fila_tot:
            fila = fila_tot.pop(0)
            channel_1.append(fila[0])
            channel_2.append(fila[1])
            channel_3.append(fila[2])
            channel_4.append(fila[3])
            channel_5.append(fila[4])
            channel_6.append(fila[5])
            channel_7.append(fila[6])
            channel_8.append(fila[7])
        for ax in axs:
            ax.relim()
            ax.autoscale_view()
```

```
plt_1.set_ydata(channel_1)
plt_2.set_ydata(channel_2)
plt_3.set_ydata(channel_3)
plt_4.set_ydata(channel_4)
plt_5.set_ydata(channel_5)
plt_6.set_ydata(channel_6)
plt_7.set_ydata(channel_7)
plt_8.set_ydata(channel_8)
```

Onde "fila_tot"é a fila recebida do processo descrito em 4.2.4.2. As variaveis *channel_1, channel_2, channel_3, channel_4, channel_5, channel_6, channel_7 e channel_8* são variáveis do tipo *deque*.

A imagem final montada para o usuário pode ser vista na figura 29. Nela são expostos os dados retirados dos oito canais de entrada da placa de captura. O estilo foi mantido simples mas pode ser alterado de acordo com os parâmetros passados nas variaveis *line_style_n*, sendo "n"o canal em que se deseja alterar o estilo da linha do mesmo. As escalas podem ser modificadas de acordo com os parâmetros *p_min*, *p_max* e *p_passos*. Caso estes parâmetros sejam retirados e as linhas com *set_ylim(...)* e *set_yticks(...)* sejam comentadas, o gráfico irá se auto ajustar ao sinal que está sendo exposto na tela. Isto é muito útil ao buscar o sinal correto, porém, pode trazer uma queda na performance do código.

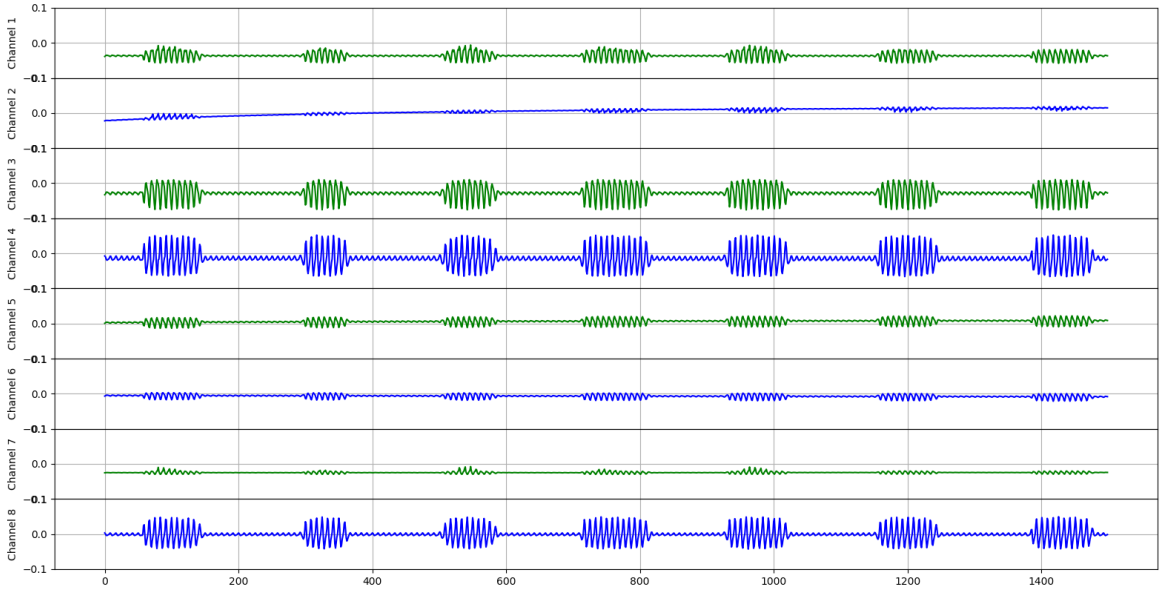

Figura 29 – Janela gráfica do programa – Fonte: Autor.

5 RESULTADOS E DISCUSSÃO

5.1 *HARDWARE*

Após todos os estudos realizados para que ocorresse a correta seleção do CI principal, foi escolhido o ADS1298 devido aos seus 24 *bits* de quantização em seus ADCs, ter uma taxa de transferência de dados suficientemente alta para suprir o sinal que será inserido em sua entrada, podendo chegar até a 4k SPS por canal, seu alto CMRR, de -115 dB, e ganho suficientemente alto para trabalhar com o sinal externamente.

Tendo selecionado todos os componentes que foram utilizados, a placa final foi construída e pode ser observado na figura 30. Nesta figura, encontram-se duas placas, uma com a frente e a outra com o verso exposto. Foi optado por utilizar uma máscara acima das trilhas para dar um melhor acabamento a placa e gerar uma proteção contra danos climáticos.

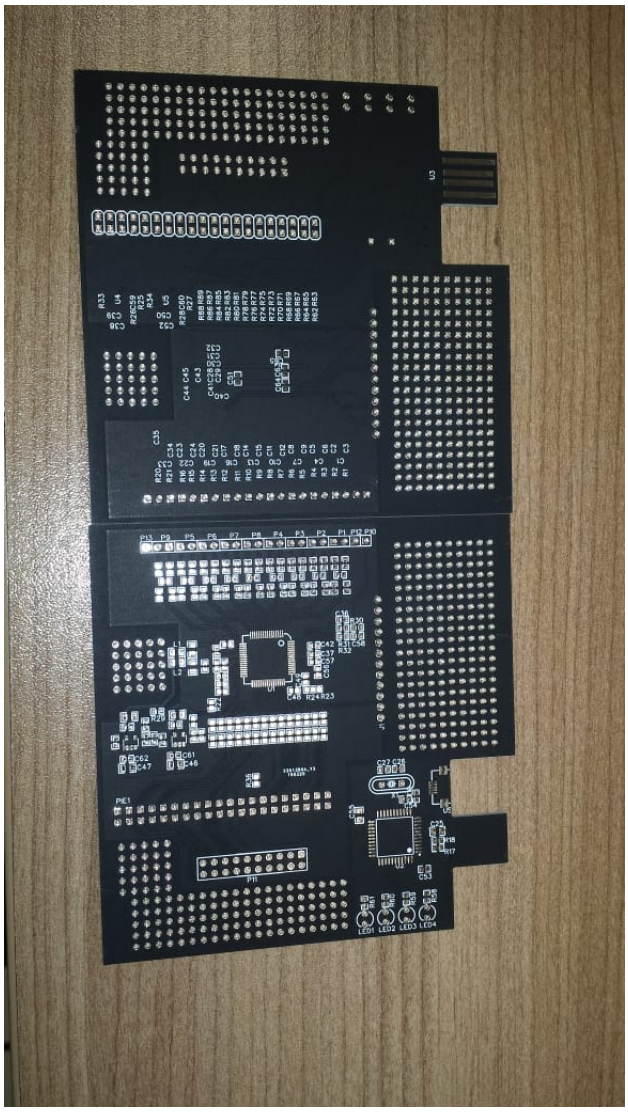

Figura 30 – Placas finais do projeto – Fonte: Autor.

Após a compra dos componentes, a placa teve seus componentes soldados e foi montada, resultando no estado exposto na figura 31.

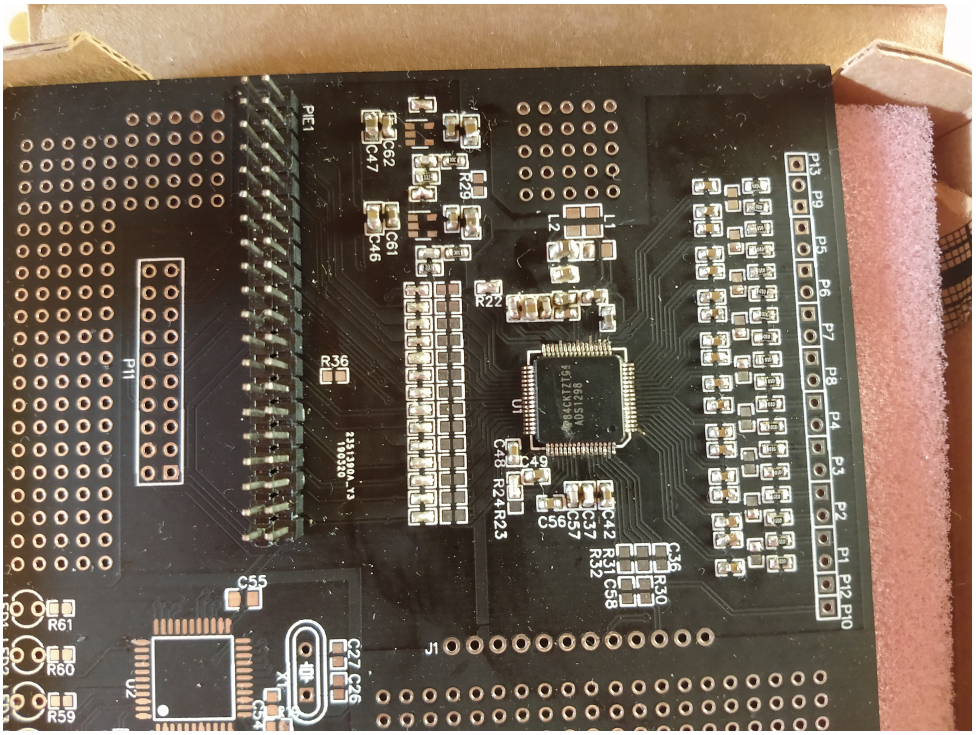

Figura 31 – Placa de aquisição soldada – Fonte: Autor.

Quando a placa foi montada foi notado um outro problema, o conector entre os eletrodos e os terminais de entrada da placa não estava disponível no mercado.

Como solução, um conector para este fim foi projetado e impresso em uma impressora 3D. O conector pode ser observado nas figuras 32 e 33.

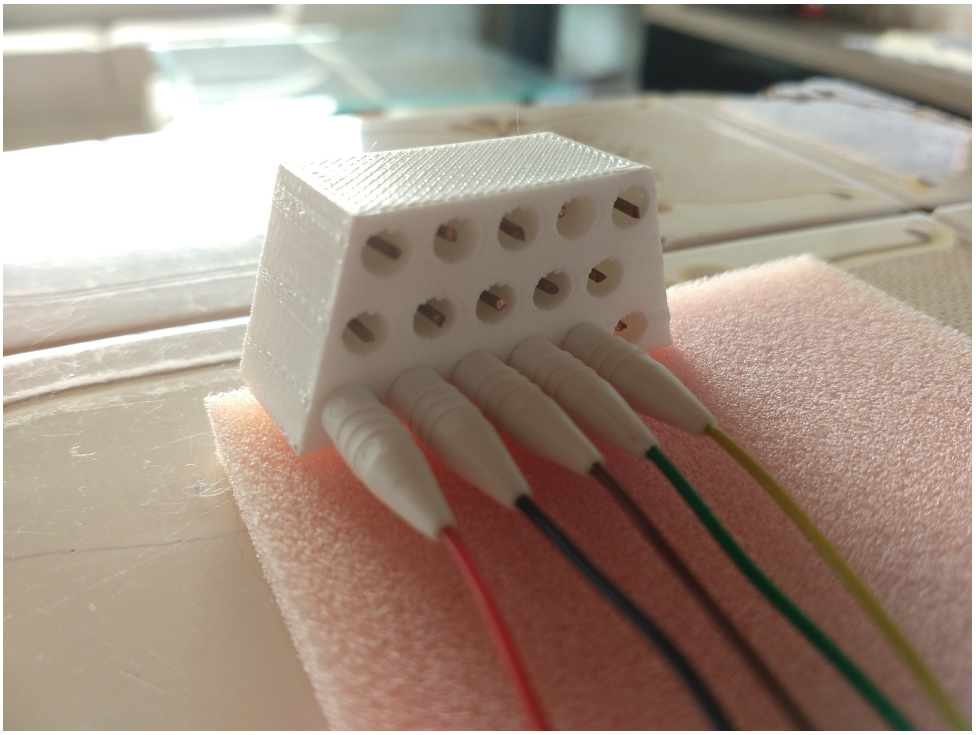

Figura 32 – Conector impresso. Vista frontal. – Fonte: Autor.

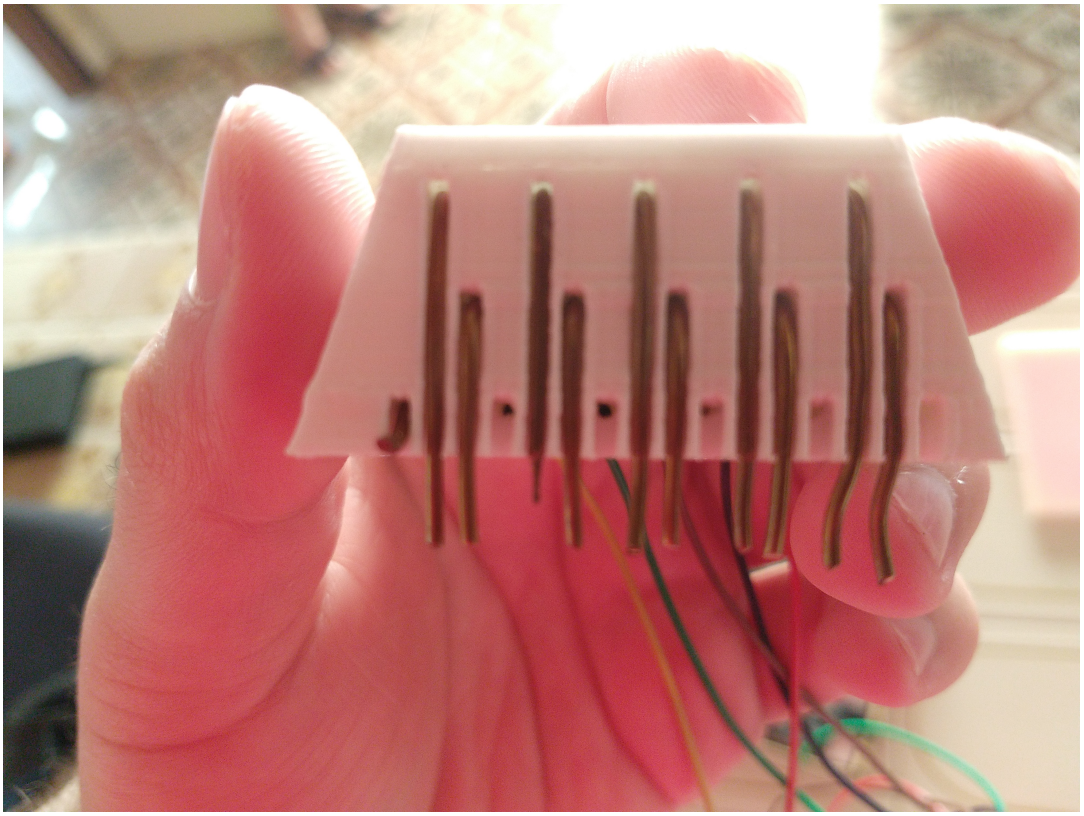

Figura 33 – Conector impresso. Vista traseira. – Fonte: Autor.

Na figura 34 pode ser visto a placa após a solda do conector nos seus terminais de saída, pronta para ser testada.

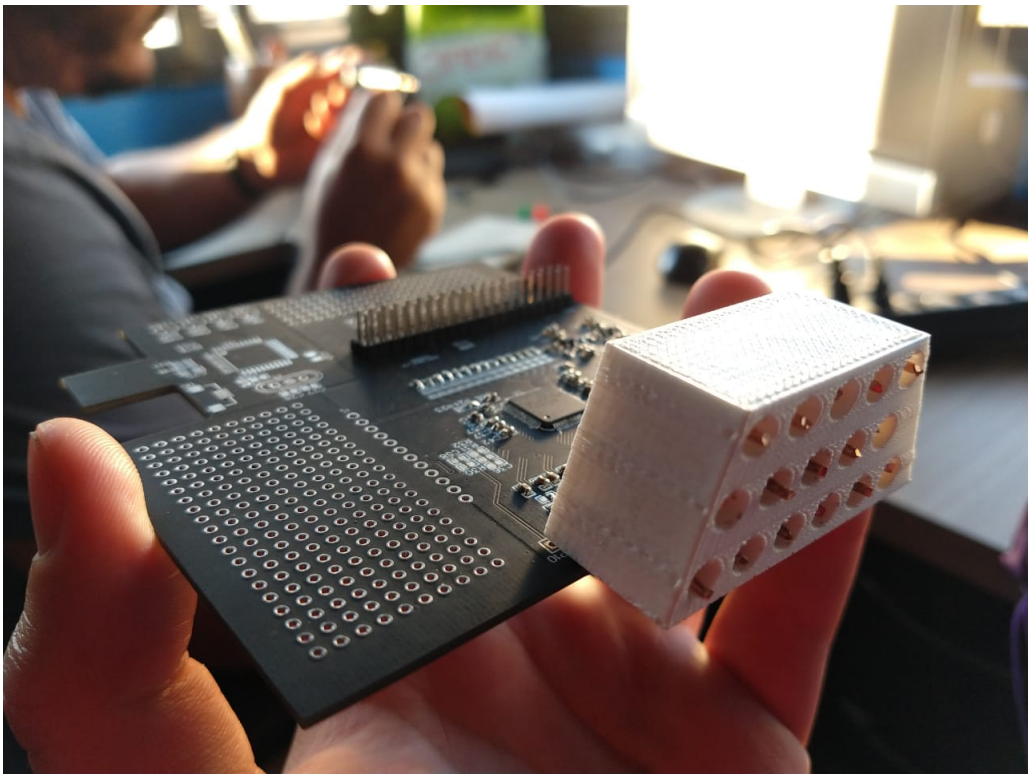

Figura 34 – Conector impresso. Vista traseira. – Fonte: Autor.

5.2 *SOFTWARE*

Com a parte de *hardware* finalizada foi passado para a parte de *software*. O primeiro passo foi realizar com sucesso a comunicação do arduino com o computador. Ao arduíno foi deixado apenas a tarefa de escutar a SPI e converter para o padrão USB, a fim de não carrega-lo com muito processamento.

Concluindo a parte de comunicação com o computador foi implementada a leitura da USB, conforme descrito em 4.2.4.1, os filtros digitais, descritos em 4.2.4.2 e a parte gráfica conforme descrito na seção 4.2.4.3.

5.3 TESTES

5.3.1 Teste com tensão de referência interna.

Para simular o funcionamento do CI, o ADS1298 dispõe de uma função capaz de emular um sinal interno, permitindo assim realizar um teste de todas as funções após a entrada, no caso, o SPI e a comunicação via USB.

Este sinal gerado é uma onda quadrada com 10m V de amplitude com 0.5 Hz de frequência. O mesmo pode ser visto em 35.

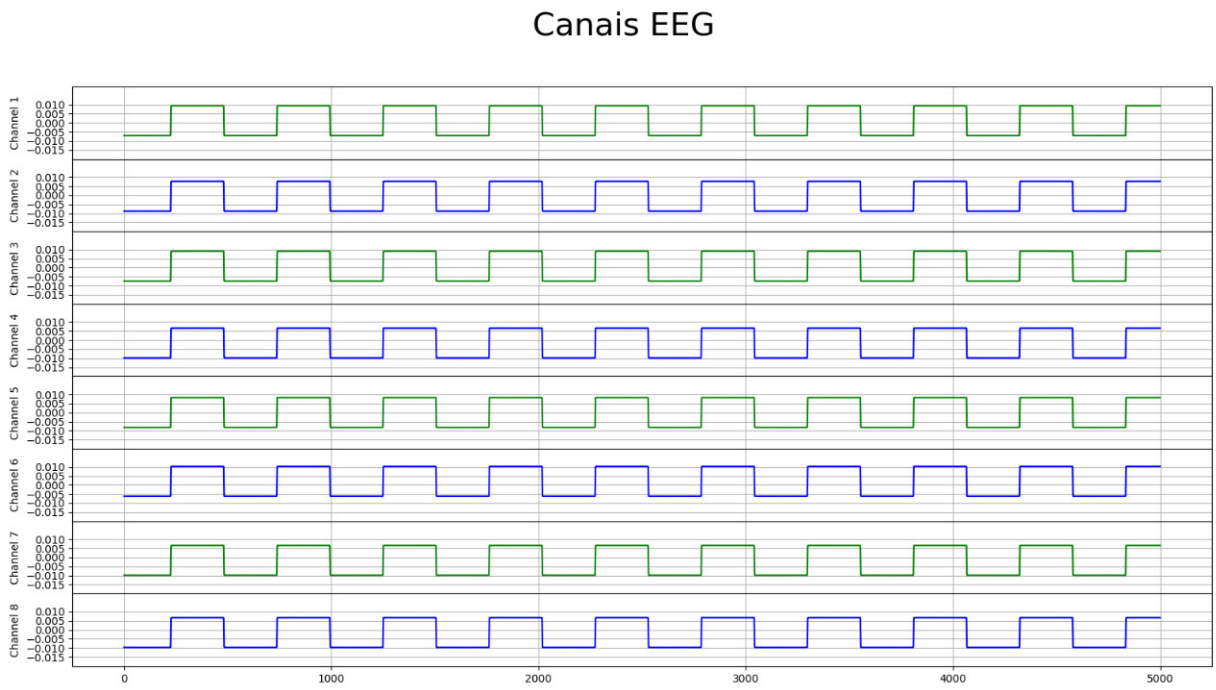

Figura 35 – Onda senoidal de 2m Vpp. – Fonte: Autor.

Como este teste foi bem sucedido, pode-se garantir que a comunicação está funcionando corretamente entre a placa e o computador.

5.3.2 Teste em um laboratório de medições.

Para garantir o correto funcionamento da placa a mesma foi levada a um laboratório de testes, neste laboratório foi gerado uma tensão senoidal com amplitude de 2mVpp e frequência de 40 Hz e foi colocado a placa desenvolvida para ler esta onda. O resultado do teste pode ser visto na figura 36.

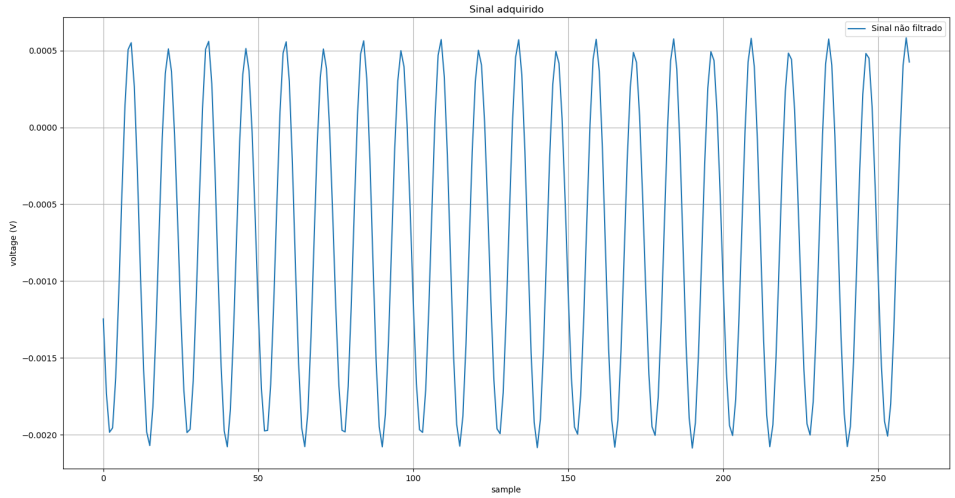

Figura 36 – Onda senoidal de 2m Vpp. – Fonte: Autor.

Conforme observado na figura 36 o teste realizado foi muito bem sucedido pois, além de a onda ter sido capturada corretamente, a mesma apresenta a amplitude desejada para o mesmo. Isto demonstra que a placa de aquisição de sinais funciona corretamente e sem injetar ruídos desta magnitude no sinal obtido.

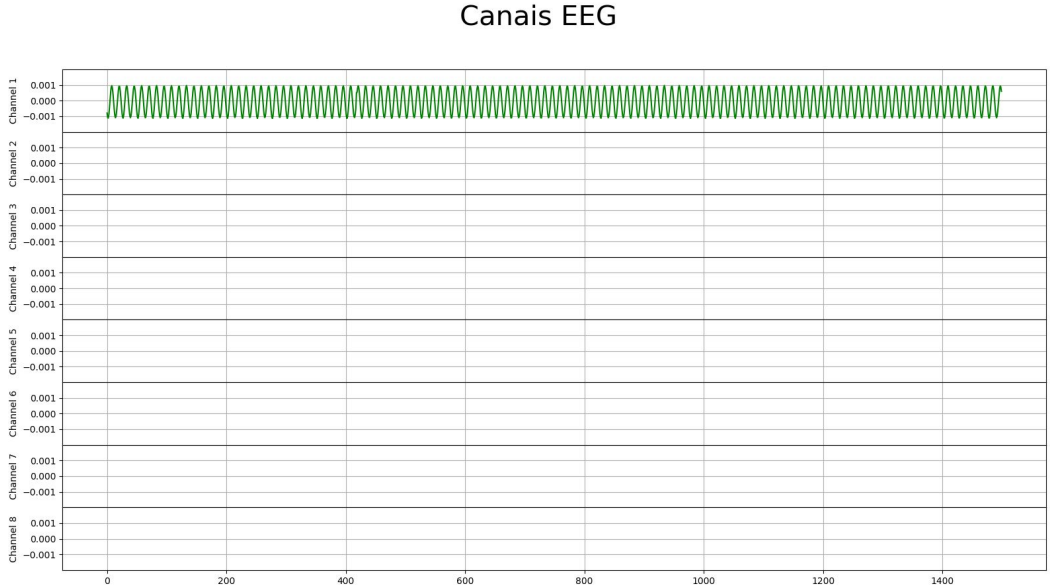

Figura 37 – Onda senoidal de 2m Vpp com filtro. – Fonte: Autor.

Esta captura foi realizada sem a utilização do código do filtro, para saber se o *hardware* não estava causando nenhuma distorção no sinal.

Após a implementação do filtro, foi utilizado o mesmo sinal gerado por este gerador para calibrar a compensação pelo uso do filtro em termos de amplitude. Esta compensação ficou na ordem de 5 vezes a magnitude de resposta do filtro de blackman. O resultado deste teste pode ser visto em 37.

5.3.3 Teste em um indivíduo.

Com todas as partes prontas e funcionais foi possivel testar a placa em um individuo. Para isto, os eletrodos foram posicionados conforme especificado em 3.2.2. Este posicionamento pode ser visto nas figuras 38, 39, 40 e 41.

Figura 38 – Cabeça com eletrodos posicionados. Vista lateral direita. – Fonte: Autor.

Figura 39 – Cabeça com eletrodos posicionados. Vista superior. – Fonte: Autor.

Figura 40 – Cabeça com eletrodos posicionados. Vista traseira. – Fonte: Autor.

Figura 41 – Cabeça com eletrodos posicionados. Vista frontal. – Fonte: Autor.

Para esta parte recomenda-se o uso de pasta em excesso, a fim de evitar qualquer mal contato que possa ocorrer entre a superfície da pele do individuo e o eletrodo em si.

O posicionamento foi realizado visando uma distribuição simétrica dos eletrodos através da superfície crâniana do indivíduo.

Com os eletrodos corretamente posicionados foi iniciado o processo de captura do sinal. O sinal de EEG capturado com a utilização deste *software* pode ser observada nas figuras 42, 43, 44 e 45.

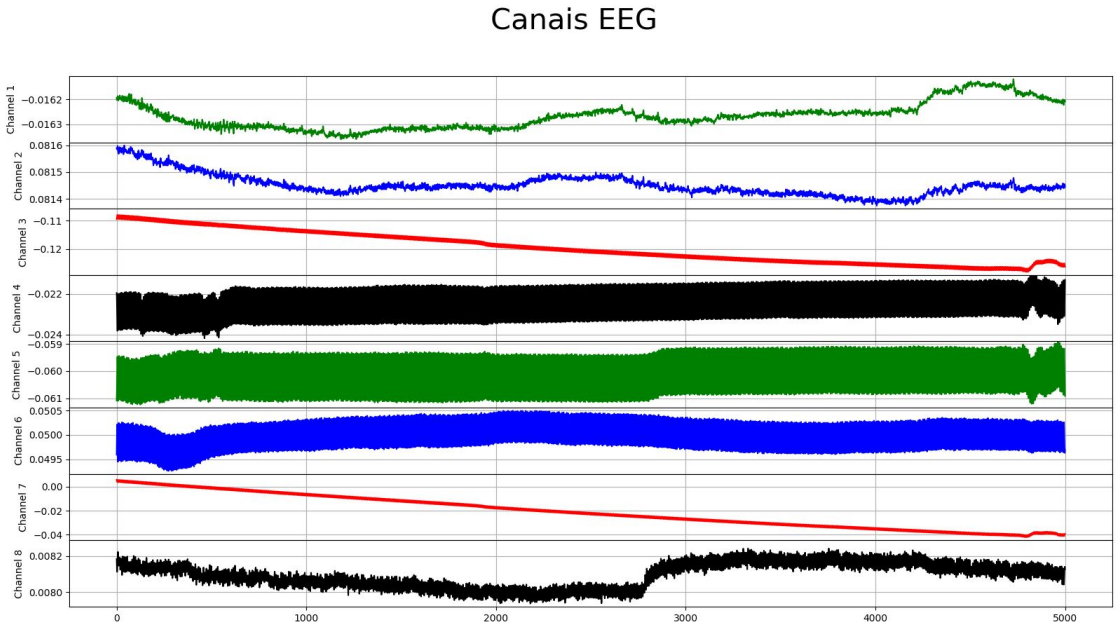

Figura 42 – Sinal extraído através do EEG. – Fonte: Autor.

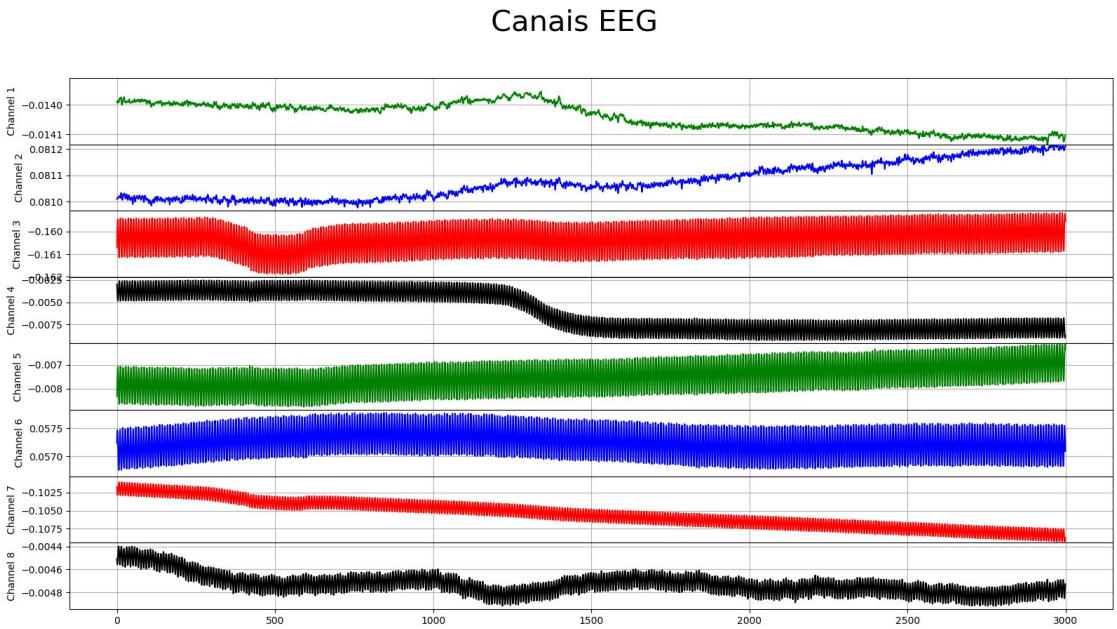

Figura 43 – Sinal extraído através do EEG. – Fonte: Autor.

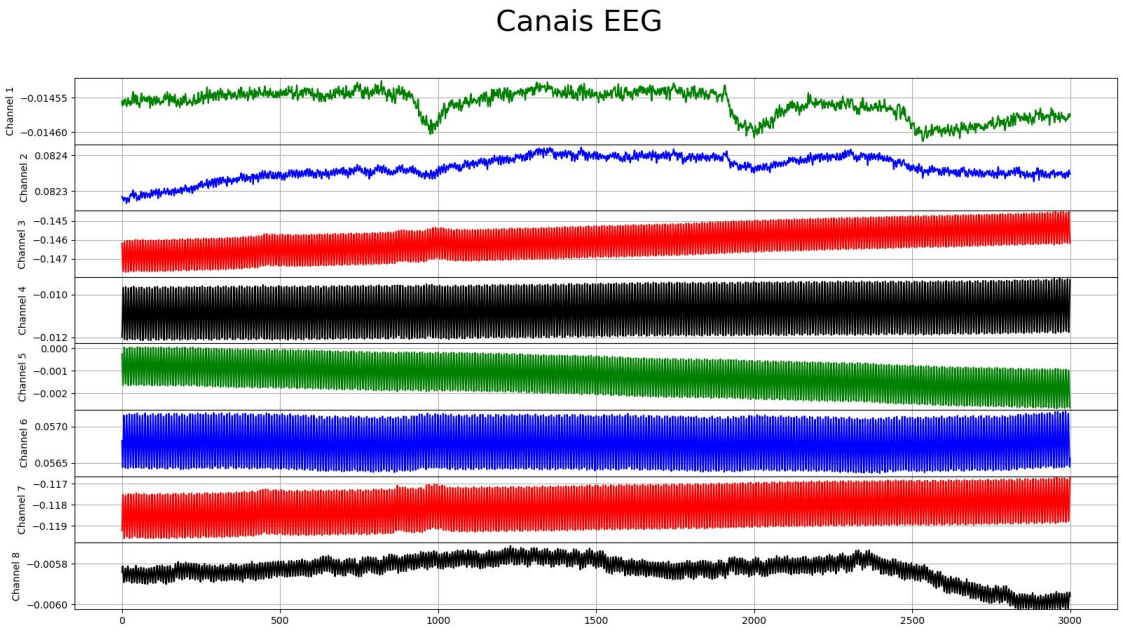

Figura 44 – Sinal extraído através do EEG. – Fonte: Autor.

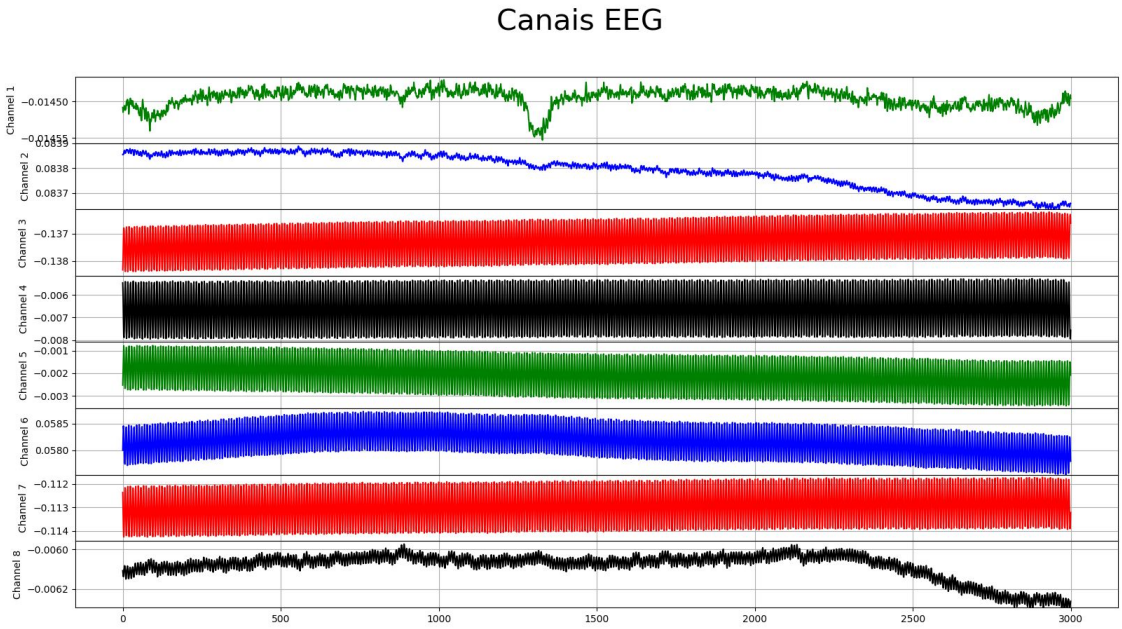

Figura 45 – Sinal extraído através do EEG. – Fonte: Autor.

Após analisar as 4 imagens, exibidas em 42, 43, 44 e 45, durante a execução do teste, notou-se que, por mais que os canais 1 e 2 estavam obtendo um sinal com todas as caracteristicas de um sinal de EEG, os demais canais aparentavam estar recebendo mais ruído do que traços do sinal em si.

Tendo constado isto, os demais eletrodos foram retirados e recolocados no mesmo local, porém mais perto da superfície crâniana do indivíduo e com menos pasta entre eles e a pele. Também foi trocada a massa externa a fim de aumentar a fixação dos eletrodos na cabeça do indivíduo.

O resultado desta realocação pode ser visto na figura 46. Nesta figura nota-se que o sinal ficou muito mais característico nos demais canais com um sinal padrão de EEG do que nas figuras 42, 43, 44 e 45.

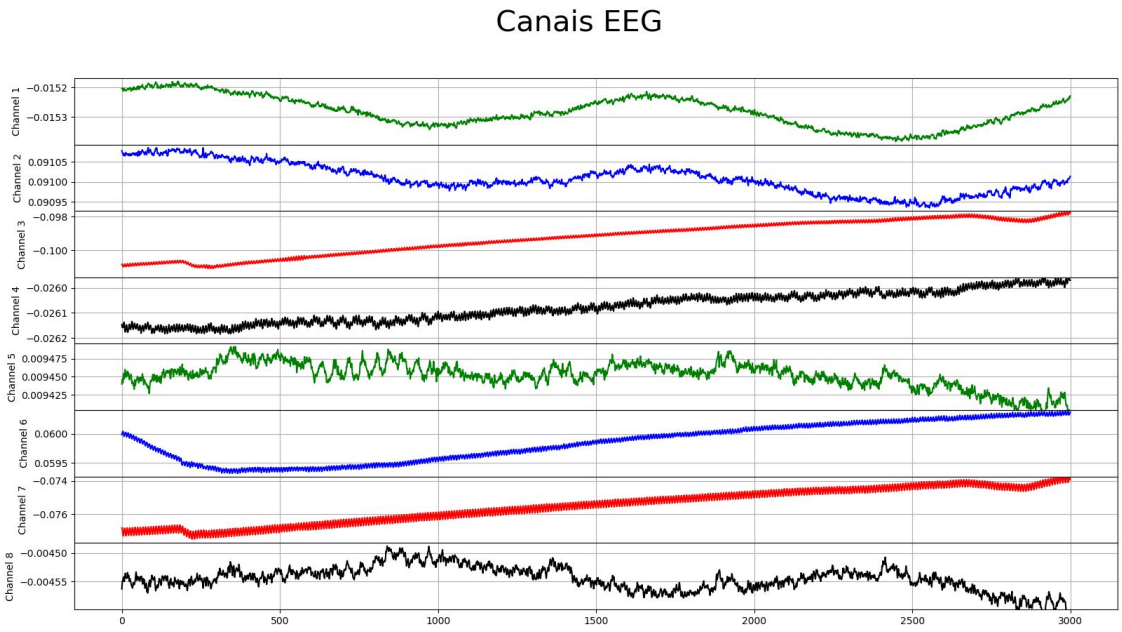

Figura 46 – Sinal extraído através do EEG. – Fonte: Autor.

6 CONCLUSÃO

O trabalho se iniciou pela seleção dos componentes que foram utilizados juntamente com o processo de confecção da placa de aquisição de sinais de EEG.

A etapa de seleção dos componentes foi bem sucedida, tendo em vista que todos os parâmetros necessários e propostos inicialmente foram cobertos pelo *hardware* disponível no mercado.

Quanto a etapa de confecção e importação da placa e dos componentes, esta também foi bem sucedida, pois não ocorreu nenhum imprevisto no processo de construção da placa nem no processo de importação dos componentes. Quando a placa chegou foi iniciado o processo de soldagem dos componentes na placa, que também foi bem positiva.

Após o encerramento da montagem da placa, foi implementado a interface SPI para a retirada dos dados da placa. Nesta etapa infelizmente foi detectado uma falha na especificação inicial do trabalho, a placa Raspberry Pi, enquanto utilizar o sistema operacional Raspbean, não é capaz de fornecer o necessário para realizar a comunicação com o SPI da placa, pois este sistema possuí como resposta minima o tempo de 1 milisegundo, o que é muito acima do tempo mínimo de resposta necessário para a interface do SPI. A fim de transpassar isto, foi tentado deixar o *hardware* da Raspberry Pi responsável pela geração do *clock* da SPI e também por gerar interrupções quando fosse possivel realizar uma leitura da placa. Estas duas melhorias trouxeram um ganho de performance para a placa, porém, mesmo após isto a placa cometia equívocos na hora de interpretar os *bits* de resposta e acabava se perdendo tanto na hora de configurar o ADS1298 quanto na hora de receber as leituras.

Para solucionar o problema encontrado, foi passado para um microcontrolador da familia atmega, que não carregava um sistema operacional. Desta forma foi possível realizar a interface SPI e ainda converter o formato de saída da placa para um USB padrão, para compatibilizar com qualquer dispositivo capaz de ler esta entrada.

Encerrando a parte de comunicação entre as placas foi realizado o processo de desenvolvimento com a linguagem python do programa descrito em 4.2.4. Este processo seguiu sem grandes problemas, a linguagem python além de possibilitar a implementação fácil das mais diversas funções também permite otimizar o código para ganhar agilidade na resolução de problemas, o que permite a implementação de novas funcionalidades no futuro.

Com todas as partes prontas, a placa foi testada em um laboratório com um sinal senoidal de *input* de amplitude de 2mVpp. O resultado deste teste pode ser visto em 36. Conforme esperado, a placa demonstrou que é capaz de obter sinais nesta magnitude sem afeta-los.

Após o teste em laboratório, a placa foi submetida ao teste de medição de sinais de EEG em um indivíduo, como pode ser observado em 5.3.3. Este teste exigia mais da placa pois a mesma precisava capturar sinais menores do que os testados na mesma. Porém, após a correta configuração dos eletrodos na cabeça do indivíduo a captura ocorreu com sucesso, demonstrando que a placa é capaz de trabalhar com estes sinais.

REFERÊNCIAS

ALCANTARA, F. M. **Monitoramento de EEG**. [S.l.], 2006. Disponível em: <https://www.up.edu.br/blogs/engenharia-da-computacao/wp-content/uploads/sites/6/2015/06/2006.13.pdf>. Acesso em: 14/06/2019. Citado na página 18.

ARNDT, D. M. **Filtragem analógica**. [S.l.], 2016. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/images/d/d6/FiltroAnalogico2.pdf>. Acesso em: 29/08/2019. Citado 3 vezes nas páginas 7, 21 e 22.

BOTUCATU, H. das Clínicas da Faculdade de Medicina de. **Conceitos básicos em EEG.** UNESP, 2015. Disponível em: <www.hcfmb.unesp.br/wp-content/uploads/2015/02/Conceitos-básicos-nov2015.pdf>. Acesso em: 28/04/2019. Citado 2 vezes nas páginas 7 e 18.

CABOCLO, L. otavio. **Bases da eletroencefalografia**. [S.l.], 2013. Disponível em: <http://www.itarget.com.br/newclients/sbnc.org.br/arquivos/aulas/aula-5.pdf>. Acesso em: 14/06/2019. Citado na página 19.

GUYTON, J. E. H. A. C. **Tratado de Fisiologia Médica.** Rio de Janeiro., 1992. Acesso em: 29/04/2019. Citado na página 17.

INFOESCOLA. **Neurônio.** InfoEscola, 2007. Disponível em: <https://www.infoescola.com/sistema-nervoso/neuronios/>. Acesso em: 29/04/2019. Citado 2 vezes nas páginas 7 e 17.

PAVAN, A. C. T. **Introdução a filtros digitais**. USP, 2011. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/1687753/mod_resource/content/0/Filtros.pdf>. Acesso em: 14/11/2019. Citado na página 23.

POERSCH, J. M. **Simulações Conexionistas: A inteligência artificial moderna.** Tubarão - SC, 2004. Citado na página 17.

PYSCIENCE-BRASIL. **SciPy**. http://pyscience-brasil.wikidot.com/, 2010. Disponível em: <http://pyscience-brasil.wikidot.com/module:scipy>. Acesso em: 14/11/2019. Citado na página 31.

RASPBERRY. **Raspberry Pi 3 Modelo B**. [S.l.], 2018. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Acesso em: 27/03/2019. Citado 2 vezes nas páginas 7 e 27.

SANEI, S. **EEG Signal Processing**. New York - EUA, 2007. Citado 2 vezes nas páginas 7 e 20.

SUNSHINE_ART. **Funções e anatomia do cérebro humano**. [S.l.], 2015. Disponível em: <https://br.depositphotos.com/91288092/stock-illustration-human-brain-anatomy-and-functions.html>. Acesso em: 14/06/2019. Citado 2 vezes nas páginas 7 e 21.

CIRCUITS BASICS. **Basics of the SPI Communication Protocol**. [S.l.], 2016. Disponível em: <http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/>. Acesso em: 01/09/2019. Citado 3 vezes nas páginas 7, 29 e 30.

TEXAS INSTRUMENTS. **ADS1198**. [S.l.], 2019. Disponível em: <http://www.ti.com/product/ADS1198>. Acesso em: 30/04/2019. Citado 3 vezes nas páginas 7, 23 e 24.

TEXAS INSTRUMENTS. **ADS1298**. [S.l.], 2019. Disponível em: <http://www.ti.com/product/ADS1298>. Acesso em: 30/04/2019. Citado 9 vezes nas páginas 7, 24, 25, 33, 36, 37, 38, 39 e 40.

TEXAS INSTRUMENTS. **ADS1299**. [S.l.], 2019. Disponível em: <http://www.ti.com/product/ADS1299>. Acesso em: 30/04/2019. Citado 3 vezes nas páginas 7, 25 e 26.

TEXAS INSTRUMENTS. **ARM® Cortex®-M4F Based MCU TM4C123G LaunchPad™ Evaluation Kit**. [S.l.], 2019. Disponível em: <http://www.ti.com/tool/EK-TM4C123GXL>. Acesso em: 29/03/2019. Citado 3 vezes nas páginas 7, 27 e 28.

TEXAS INSTRUMENTS. **TMS320C5535 Fixed-Point Digital Signal Processor**. [S.l.], 2019. Disponível em: <http://www.ti.com/product/TMS320C5535>. Acesso em: 29/03/2019. Citado 3 vezes nas páginas 7, 28 e 29.

WEBSTER, J. G. **Medical Instrumentation - Application and Desing**. Danvers - MA, 2010. Citado 3 vezes nas páginas 7, 19 e 20.

APÊNDICE A – INFORMAÇÕES COMPLEMENTARES

A.1 CRONOGRAMA

A.1.1 Gráfico Gantt

Na figura 47 é possível ver o gráfico gantt do cronograma que foi montado para este projeto. O inicio do projeto se dá na data 05/08/2019 e se estende até a data da defesa final, no dia 05/12/2019.

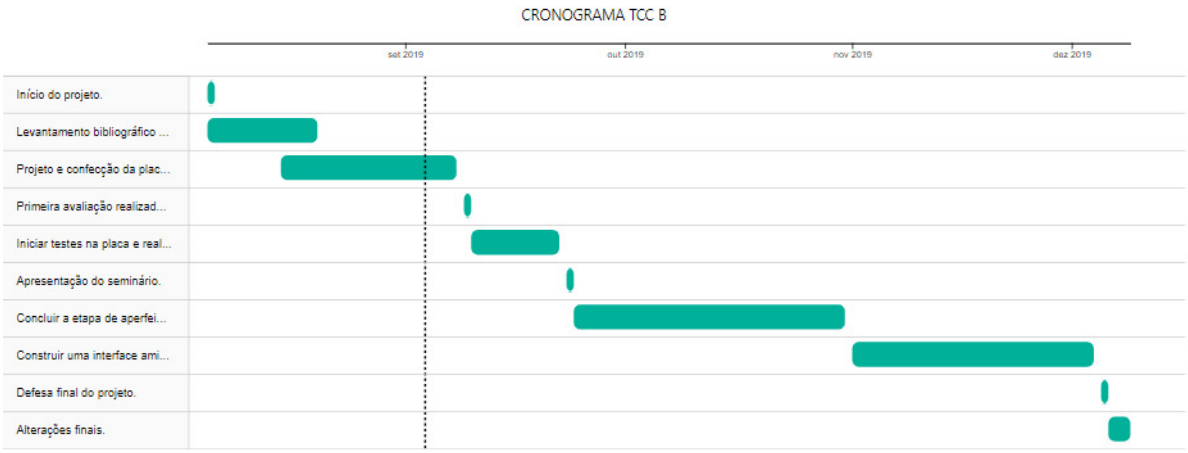

Figura 47 – Gráfico gantt do cronograma. Fonte: Autor.

Logo abaixo, na seção A.1.2 é possível observar o cronograma simplificado com a data de inicio e fim de cada etapa. Por sua vez, na seção A.1.3 se encontra um cronograma sem datas, onde cada etapa foi dividida em etapas menores e mais detalhadas.

A.1.2 Simplificado com datas

Etapa	Inicio	Fim
Início do projeto.	05/08/19	05/08/19
Levantamento bibliográfico do projeto.	05/08/19	20/08/19
Projeto e confecção da placa de aquisição de sinais.	15/08/19	08/09/19
Primeira avaliação realizada pelo orientador.	09/09/19	09/09/19
Iniciar testes na placa e realizar o processamento dos sinais adquiridos.	10/09/19	22/09/19
Apresentação do seminário.	23/09/19	23/09/19
Concluir a etapa de aperfeiçoamento do processamento de sinais.	24/09/19	31/10/19
Construir uma interface amigável para exibição dos resultados obtidos.	01/11/19	04/12/19
Defesa final do projeto.	05/12/19	05/12/19
Possiveis alterações finais.	06/12/19	09/12/19

A.1.3 Etapas detalhadas

1. Elaboração da proposta de início do projeto;

2. Apresentação da proposta de início do projeto;
3. Estudo para caracterização dos formatos de ondas de EEG;
4. Levantamento dos principais *hardwares* disponíveis para esta função no mercado;
5. Realizar o projeto da placa eletrônica para a aquisição de sinais;
6. Iniciar a montagem da placa eletrônica para a aquisição de sinais;
7. Estudos de técnicas de processamento de sinal;
8. Finalizar a montagem da placa eletrônica para a aquisição de sinais;
9. Primeira avaliação realizada pelo orientador dia 09/09/2019;
10. Iniciar testes na placa;
11. Realizar o pré-processamento dos sinais de onda a fim de melhorar o sinal obtido;
12. Avaliar a onda final obtida da placa montada;
13. Iniciar o processamento da caracterização das principais ondas de EEG;
14. Apresentação do seminário dia 23/09/2019;
15. Evoluir no processamento do sinal adquirido a fim de melhorar a caracterização das principais ondas do EEG;
16. Construir uma interface amigavel para exibição dos resultados obtidos;
17. Defesa final do projeto dia 05/12/2019.
18. Ajustes finais pós-retorno da banca.

APÊNDICE B – CÓDIGOS

B.0.1 Programa principal - Python.

```
from multiprocessing import Process, Manager
import serial
import io
from collections import deque
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import random
from scipy import signal

def consumer (fila, fila_graf):
    history_ch1 = []
    history_ch2 = []
    history_ch3 = []
    history_ch4 = []
    history_ch5 = []
    history_ch6 = []
    history_ch7 = []
    history_ch8 = []
    ordem = 50
    f = 40
    fs = 500
    black = signal.firwin((ordem+1), f, window='blackman', nyq = fs/2)
    comp = sum(black)
    while(True):
        if len(fila) > 0:
            for x in range(len(fila)):
                retorno = fila.pop(0)
                history_ch1.append(retorno[0])
                history_ch2.append(retorno[1])
                history_ch3.append(retorno[2])
                history_ch4.append(retorno[3])
                history_ch5.append(retorno[4])
                history_ch6.append(retorno[5])
                history_ch7.append(retorno[6])
                history_ch8.append(retorno[7])
```

```
            pontos_filtrados_ch1 = \
                signal.convolve(history_ch1, black, mode='same')/comp
            pontos_filtrados_ch2 = \
                signal.convolve(history_ch2, black, mode='same')/comp
            pontos_filtrados_ch3 = \
                signal.convolve(history_ch3, black, mode='same')/comp
            pontos_filtrados_ch4 = \
                signal.convolve(history_ch4, black, mode='same')/comp
            pontos_filtrados_ch5 = \
                signal.convolve(history_ch5, black, mode='same')/comp
            pontos_filtrados_ch6 = \
                signal.convolve(history_ch6, black, mode='same')/comp
            pontos_filtrados_ch7 = \
                signal.convolve(history_ch7, black, mode='same')/comp
            pontos_filtrados_ch8 = \
                signal.convolve(history_ch8, black, mode='same')/comp
            for x in range(len(pontos_filtrados_ch1)):
                linha_pos_filtro=[pontos_filtrados_ch1[x],\
                    pontos_filtrados_ch2[x],pontos_filtrados_ch3[x],\
                    pontos_filtrados_ch4[x],pontos_filtrados_ch5[x],\
                    pontos_filtrados_ch6[x],pontos_filtrados_ch7[x],\
                    pontos_filtrados_ch8[x]]
                fila_graf.append(linha_pos_filtro)
                arq = open('simulado.csv', 'a')
                arq.write(str(linha_pos_filtro).replace('[','')\
                    .replace(']','').replace(' ','')+'\n')
                arq.close()
            history_ch1.clear()
            history_ch2.clear()
            history_ch3.clear()
            history_ch4.clear()
            history_ch5.clear()
            history_ch6.clear()
            history_ch7.clear()
            history_ch8.clear()

def graf (fila_tot):
    def animate(i):
        for lec in fila_tot:
```

```
            fila = fila_tot.pop(0)
            channel_1.append(fila[0])
            channel_2.append(fila[1])
            channel_3.append(fila[2])
            channel_4.append(fila[3])
            channel_5.append(fila[4])
            channel_6.append(fila[5])
            channel_7.append(fila[6])
            channel_8.append(fila[7])
        for ax in axs:
            ax.relim()
            ax.autoscale_view()
        plt_1.set_ydata(channel_1)
        plt_2.set_ydata(channel_2)
        plt_3.set_ydata(channel_3)
        plt_4.set_ydata(channel_4)
        plt_5.set_ydata(channel_5)
        plt_6.set_ydata(channel_6)
        plt_7.set_ydata(channel_7)
        plt_8.set_ydata(channel_8)
    lim_inf_y = -0.02
    lim_max_y = 0.02
    p_max = 0.1+0.001
    p_min = -0.1
    p_passos = 0.1
    window_size = 1500
    line_style_1 = 'g-'
    line_style_2 = 'b-'
    line_style_3 = 'g-'
    line_style_4 = 'b-'
    line_style_5 = 'g-'
    line_style_6 = 'b-'
    line_style_7 = 'g-'
    line_style_8 = 'b-'
    grid_on = True

    channel_1 = deque([0]*window_size, maxlen=window_size)
    channel_2 = deque([0]*window_size, maxlen=window_size)
    channel_3 = deque([0]*window_size, maxlen=window_size)
```

```
channel_4 = deque([0]*window_size, maxlen=window_size)
channel_5 = deque([0]*window_size, maxlen=window_size)
channel_6 = deque([0]*window_size, maxlen=window_size)
channel_7 = deque([0]*window_size, maxlen=window_size)
channel_8 = deque([0]*window_size, maxlen=window_size)
fig = plt.figure()
axs = fig.subplots(8, 1, sharex=True)
fig.subplots_adjust(hspace=0)
fig.suptitle('Canais EEG', fontsize=30)

plt_1, = axs[0].plot(channel_1, line_style_1, label='Channel 1')
axs[0].set_ylim(lim_inf_y,lim_max_y)
axs[0].set_ylabel('Channel 1')
axs[0].grid(grid_on)
axs[0].set_yticks(np.arange(p_min, p_max, p_passos))

plt_2, = axs[1].plot(channel_2, line_style_2, label='Channel 2')
axs[1].set_ylim(lim_inf_y,lim_max_y)
axs[1].set_ylabel('Channel 2')
axs[1].grid(grid_on)
axs[1].set_yticks(np.arange(p_min, p_max, p_passos))

plt_3, = axs[2].plot(channel_3, line_style_3, label='Channel 3')
axs[2].set_ylim(lim_inf_y,lim_max_y)
axs[2].set_ylabel('Channel 3')
axs[2].grid(grid_on)
axs[2].set_yticks(np.arange(p_min, p_max, p_passos))

plt_4, = axs[3].plot(channel_4, line_style_4, label='Channel 4')
axs[3].set_ylim(lim_inf_y,lim_max_y)
axs[3].set_ylabel('Channel 4')
axs[3].grid(grid_on)
axs[3].set_yticks(np.arange(p_min, p_max, p_passos))

plt_5, = axs[4].plot(channel_5, line_style_5, label='Channel 5')
axs[4].set_ylim(lim_inf_y,lim_max_y)
axs[4].set_ylabel('Channel 5')
axs[4].grid(grid_on)
axs[4].set_yticks(np.arange(p_min, p_max, p_passos))
```

```
    plt_6, = axs[5].plot(channel_6, line_style_6, label='Channel 6')
    axs[5].set_ylim(lim_inf_y,lim_max_y)
    axs[5].set_ylabel('Channel 6')
    axs[5].grid(grid_on)
    axs[5].set_yticks(np.arange(p_min, p_max, p_passos))

    plt_7, = axs[6].plot(channel_7, line_style_7, label='Channel 7')
    axs[6].set_ylim(lim_inf_y,lim_max_y)
    axs[6].set_ylabel('Channel 7')
    axs[6].grid(grid_on)
    axs[6].set_yticks(np.arange(p_min, p_max, p_passos))

    plt_8, = axs[7].plot(channel_8, line_style_8, label='Channel 8')
    axs[7].set_ylim(lim_inf_y,lim_max_y)
    axs[7].set_ylabel('Channel 8')
    axs[7].grid(grid_on)
    axs[7].set_yticks(np.arange(p_min, p_max, p_passos))

    ani = animation.FuncAnimation(fig, animate, interval=1)
    plt.show()

def productor(fila):
    eegReading = [0]*8
    ser = serial.Serial('COM4',1000000,timeout=1)
    if(ser.is_open):
        str(ser.readline())
        while(True):
            line = str(ser.readline()).replace('b\'','')
            lineRead = line.split(',')
            for i in range(8):
                lineRead[i] = int(lineRead[i],16)
            for i in range(8):
                if lineRead[i] > 0x800000:
                    lineRead[i] -= 0x1000000
                eegReading[i] = lineRead[i]*3.3/0x800000
            fila.append(eegReading)
    ser.close()
```

```
if __name__ == '__main__':
    with Manager() as manager:
        d = manager.dict()
        history = manager.list()
        history_graf = manager.list()
        prod = Process(target=productor, args=(history,))
        prod.start()
        cons = Process(target=consumer, args=(history,history_graf))
        cons.start()
        cons_graf = Process(target=graf, args=(history_graf,))
        cons_graf.start()
        cons_graf.join()
```

B.0.2 Código principal do arduíno - ADS1298.

```
void setup() {
  //Inicialização do USB
  Serial.begin(1000000);

  GPIO_startup();
  SPI_startup();
  ADS_hardReset();
  //Comando interno ADS
  command(SDATAC);
  command(STOP);
  //Confirmar que é a placa com o ADS1298 de slave
  if(readReg(ID)!=0x92){
    while(1){
      delay(2000);
      Serial.println("Invalid device ID");
    }
  }
  //Initial configuration
  writeReg(CONFIG1,
    (1<<7)|^^I^^I//Modo de alta resolução
    (6)|^^I^^I//500 SPS
    0);^^I^^I^^I//Daisy chain, Osc Out disable
  writeReg(CONFIG2,
    (1<<4)|^^I^^I//Sinais de testes referenciados internamente
    (0)|^^I^^I//Pulsed at fClk/2^21
```

```
    0);^^I^^I^^I//Amplitude de teste em VRef/2400
  writeReg(CONFIG3,
    (1<<7)|^^I^^I//Buffer de refencia interna ativo
    (1<<6)|^^I^^I//Definido 1 no datasheet
    0);^^I^^I^^I//VRef at 2.4V, RLD open, RLDREF external, RLD powerdown, RLD off,
  writeReg(CH1SET, (1<<4)| 0);
  writeReg(CH2SET, (1<<4)| 0);
  writeReg(CH3SET, (1<<4)| 0);
  writeReg(CH4SET, (1<<4)| 0);
  writeReg(CH5SET, (1<<4)| 0);
  writeReg(CH6SET, (1<<4)| 0);
  writeReg(CH7SET, (1<<4)| 0);
  writeReg(CH8SET, (1<<4)| 0);

  command(START);
}

void loop() {
  while(digitalRead(DRDY_PIN));
  digitalWrite(LED0_PIN,     HIGH);
  readConversion();
  int i = 0;
  for(i = 1; i<9;i++){
    Serial.print(eegLastReading[i], HEX);
^^ISerial.print(',',HEX);^^I^^I//Formato CSV
  }
  Serial.println();
  digitalWrite(LED0_PIN,     LOW);
}
```

B.0.3 Rotina de inicialização - ADS1298.

```
#ifndef _EEG__HAL_H
#define _EEG__HAL_H

#include <SPI.h>

#define PWDN_PIN        10
#define RESET_PIN       13
#define CLK_PIN         5
```

```
#define START_PIN      6
#define DAISY_IN_PIN   12
#define AD_GPIO1_PIN   4
#define DRDY_PIN       1
#define AD_GPIO4_PIN   0
#define AD_GPIO3_PIN   2
#define AD_GPIO2_PIN   3
#define CS_PIN         11
#define LED0_PIN       A4
#define LED1_PIN       A5

#define SPI_START()          digitalWrite(CS_PIN,LOW)
#define SPI_STOP()           digitalWrite(CS_PIN,HIGH)
#define SPI_TRANSFER1(b)     SPI.transfer(b)
#define SPI_TRANSFERN(a,s)   SPI.transfer(a,s)

void SPI_startup(){
  SPI.begin();
  SPI.beginTransaction(SPISettings(1945000, MSBFIRST, SPI_MODE1));
}

void GPIO_startup(){
  pinMode(PWDN_PIN,     OUTPUT);
  pinMode(RESET_PIN,    OUTPUT);
  pinMode(CLK_PIN,      INPUT);
  pinMode(START_PIN,    OUTPUT);
  pinMode(DAISY_IN_PIN, OUTPUT);
  pinMode(DRDY_PIN,     INPUT);
  pinMode(CS_PIN,       OUTPUT);
  pinMode(AD_GPIO1_PIN, INPUT);
  pinMode(AD_GPIO2_PIN, INPUT);
  pinMode(AD_GPIO3_PIN, INPUT);
  pinMode(AD_GPIO4_PIN, INPUT);
  pinMode(LED0_PIN,     OUTPUT);
  pinMode(LED1_PIN,     OUTPUT);
  digitalWrite(PWDN_PIN,     HIGH);
  digitalWrite(RESET_PIN,    HIGH);
  digitalWrite(START_PIN,    LOW);
  digitalWrite(DAISY_IN_PIN, LOW);
```

```
  digitalWrite(CS_PIN,       HIGH);
  digitalWrite(AD_GPIO1_PIN, LOW);
  digitalWrite(AD_GPIO2_PIN, LOW);
  digitalWrite(AD_GPIO3_PIN, LOW);
  digitalWrite(AD_GPIO4_PIN, LOW);
  digitalWrite(LED0_PIN,     LOW);
  digitalWrite(LED1_PIN,     LOW);
}

void ADS_hardReset(){
  delay(150);
  digitalWrite(RESET_PIN, LOW);
  delay(1);
  digitalWrite(RESET_PIN, HIGH);
  delay(1);
}

#endif
```

B.0.4 Comandos - ADS1298.

```
#ifndef _EEG__BASIC_COMMANDS_H
#define _EEG__BASIC_COMMANDS_H
#include "hal.h"

enum Commands : uint8_t {
  //SYSTEM COMMANDS
  WAKEUP=0x02, // Wakeup from standby mode
  STANDBY=0x04, // Enter standby mode
  RESET=0x06, // Reset the device
  START=0x08, // Start/restart (synchronize) conversions
  STOP=0xA0, // Stop conversion
  //DATA READ COMMANDS
  RDATAC=0x10, // Enable Read Data Continuous mode. (default)
  SDATAC=0x11, // Stop Read Data Continuously mode
  RDATA=0x12, // Read data by command; supports multiple read back.
  //REGISTER READ COMMANDS
  //These commands work by sending 2 bytes of command:
  // (CMD | (REG_MASK & start_addr)) ; (REG_MASK & desired_count)
  REG_MASK=0x1F,
```

```
  RREG=0x20, // Read registers
  WREG=0x40, // Write registers
};

enum Registers : uint8_t {
  // DEVICE SETTINGS (READ-ONLY REGISTERS)
  ID=0x00,
  // GLOBAL SETTINGS ACROSS CHANNELS
  CONFIG1=0x01,
  CONFIG2=0x02,
  CONFIG3=0x03,
  LOFF=0x04,
  //CHANNEL-SPECIFIC SETTINGS
  CH1SET=0x05,
  CH2SET=0x06,
  CH3SET=0x07,
  CH4SET=0x08,
  CH5SET=0x09,
  CH6SET=0x0A,
  CH7SET=0x0B,
  CH8SET=0x0C,
  RLD_SENSP=0x0D,
  RLD_SENSN=0x0E,
  LOFF_SENSP=0x0F,
  LOFF_SENSN=0x10,
  LOFF_FLIP=0x11,
  //LEAD-OFF STATUS REGISTERS (READ-ONLY REGISTERS)
  LOFF_STATP=0x12,
  LOFF_STATN=0x13,
  //GPIO AND OTHER REGISTERS
  GPIO=0x14,
  PACE=0x15,
  RESP=0x16,
  CONFIG4=0x17,
  WCT1=0x18,
  WCT2=0x19,
  MAX_REG_COUNT
};
```

```
uint8_t readReg(Registers reg){
    uint8_t ret;
    SPI_START();
    SPI_TRANSFER1(RREG | reg);
    SPI_TRANSFER1(0);
    ret = SPI_TRANSFER1(0);
    SPI_STOP();
    return ret;
}

void writeReg(Registers reg, uint8_t val){
    uint8_t ret;
    SPI_START();
    SPI_TRANSFER1(WREG | reg);
    SPI_TRANSFER1(0);
    SPI_TRANSFER1(val);
    SPI_STOP();
    return ret;
}

void command(Commands c){
    SPI_START();
    SPI_TRANSFER1(c);
    SPI_STOP();
}

int32_t eegLastReading[9];
void readConversion(){
    SPI_START();
    SPI_TRANSFER1(RDATA);
    for(int i=0; i<9; i++){
      eegLastReading[i] = ((uint32_t)SPI_TRANSFER1(0))<<16;
      eegLastReading[i] |= ((uint32_t)SPI_TRANSFER1(0))<<8;
      eegLastReading[i] |= ((uint32_t)SPI_TRANSFER1(0))<<0;
      if(eegLastReading[i] & (1UL<<23))eegLastReading[i]|=0xFF000000UL;
    }
    SPI_STOP();
}
```

```
#endif
```

B.0.5 Programa principal - Raspberry Pi.

```
from multiprocessing import Process, Manager
import os, time
import RPi.GPIO as GPIO
from comands import *
from inicializer import *

def consumer (fila):
    history = []
    while(True):
        if len(fila) > 0:
            retorno = fila.pop()
            history.append(retorno)
            print(retorno)
            arq = open('simulado.csv', 'a')
            arq.write(str(retorno).replace('[','') \
                .replace(']','').replace(' ','')+'\n')
            arq.close()

def graf(fila_tot):
    def animate(i):
        fila = fila_tot.pop(0)
        channel_1.append(fila[1])
        channel_2.append(fila[2])
        channel_3.append(fila[3])
        channel_4.append(fila[4])
        channel_5.append(fila[5])
        channel_6.append(fila[6])
        channel_7.append(fila[7])
        channel_8.append(fila[8])
        for ax in axs:
            ax.relim()
            ax.autoscale_view()
        plt_1.set_ydata(channel_1)
        plt_2.set_ydata(channel_2)
        plt_3.set_ydata(channel_3)
        plt_4.set_ydata(channel_4)
```

```
    plt_5.set_ydata(channel_5)
    plt_6.set_ydata(channel_6)
    plt_7.set_ydata(channel_7)
    plt_8.set_ydata(channel_8)
window_size = 500
channel_1 = deque([0]*window_size, maxlen=window_size)
channel_2 = deque([0]*window_size, maxlen=window_size)
channel_3 = deque([0]*window_size, maxlen=window_size)
channel_4 = deque([0]*window_size, maxlen=window_size)
channel_5 = deque([0]*window_size, maxlen=window_size)
channel_6 = deque([0]*window_size, maxlen=window_size)
channel_7 = deque([0]*window_size, maxlen=window_size)
channel_8 = deque([0]*window_size, maxlen=window_size)
fig = plt.figure()
axs = fig.subplots(8, 1, sharex=True)
fig.subplots_adjust(hspace=0)
fig.suptitle('Canais EEG', fontsize=12)
line_style_1 = 'g-'
line_style_2 = 'b-'
plt_1, = axs[0].plot(channel_1, line_style_1, label='Channel 1')
axs[0].set_ylim(-1.2,1.2)
axs[0].set_ylabel('Channel 1')
axs[0].grid(False)
axs[0].set_yticks(np.arange(-1,1.1, 0.5))

plt_2, = axs[1].plot(channel_2, line_style_2, label='Channel 2')
axs[1].set_ylim(-1.2,1.2)
axs[1].set_ylabel('Channel 2')
axs[1].grid(False)
axs[1].set_yticks(np.arange(-1,1.1, 0.5))

plt_3, = axs[2].plot(channel_3, line_style_1, label='Channel 3')
axs[2].set_ylim(-1.2,1.2)
axs[2].set_ylabel('Channel 3')
axs[2].grid(False)
axs[2].set_yticks(np.arange(-1,1.1, 0.5))

plt_4, = axs[3].plot(channel_4, line_style_2, label='Channel 4')
axs[3].set_ylim(-1.2,1.2)
```

```
    axs[3].set_ylabel('Channel 4')
    axs[3].grid(False)
    axs[3].set_yticks(np.arange(-1,1.1, 0.5))

    plt_5, = axs[4].plot(channel_5, line_style_1, label='Channel 5')
    axs[4].set_ylim(-1.2,1.2)
    axs[4].set_ylabel('Channel 5')
    axs[4].grid(False)
    axs[4].set_yticks(np.arange(-1,1.1, 0.5))

    plt_6, = axs[5].plot(channel_6, line_style_2, label='Channel 6')
    axs[5].set_ylim(-1.2,1.2)
    axs[5].set_ylabel('Channel 6')
    axs[5].grid(False)
    axs[5].set_yticks(np.arange(-1,1.1, 0.5))

    plt_7, = axs[6].plot(channel_7, line_style_1, label='Channel 7')
    axs[6].set_ylim(-1.2,1.2)
    axs[6].set_ylabel('Channel 7')
    axs[6].grid(False)
    axs[6].set_yticks(np.arange(-1,1.1, 0.5))

    plt_8, = axs[7].plot(channel_8, line_style_2, label='Channel 8')
    axs[7].set_ylim(-1.2,1.2)
    axs[7].set_ylabel('Channel 8')
    axs[7].grid(False)
    axs[7].set_yticks(np.arange(-1,1.1, 0.5))

    ani = animation.FuncAnimation(fig, animate, interval=10)
    plt.show()

def productor(fila):
    #Inicializar partes
    spi = SPI_startup()
    GPIO_startup()
    ADS_hardReset()
    #Configuracao inicial do ADS
    command(spi,Commands.SDATAC.value)
    command(spi,Commands.STOP.value)
```

```
    #Garantindo que é o ADS ligado ao SPI
    regDev = readReg(spi,Registers.ID.value)
    if(regDev[2]!=0x92):
     print(readReg(spi,Registers.ID.value))
     while(True):
      time.sleep(2)
      print("Invalid device ID")
    #Initial configuration
    conf_CONFIG1 = (1<<7)|(6)|(0)
    conf_CONFIG2 = (1<<4)|(0)|(0)
    conf_CONFIG3 = (1<<7)|(1<<6)|(0)
    conf_CHANNEL = (1<<7)|(0)
    writeReg(spi, Registers.CONFIG1.value, conf_CONFIG1)
    writeReg(spi, Registers.CONFIG2.value, conf_CONFIG2)
    writeReg(spi, Registers.CONFIG3.value, conf_CONFIG3)
    writeReg(spi, Registers.CH1SET.value, conf_CHANNEL)
    writeReg(spi, Registers.CH2SET.value, conf_CHANNEL)
    writeReg(spi, Registers.CH3SET.value, conf_CHANNEL)
    writeReg(spi, Registers.CH4SET.value, conf_CHANNEL)
    writeReg(spi, Registers.CH5SET.value, conf_CHANNEL)
    writeReg(spi, Registers.CH6SET.value, conf_CHANNEL)
    writeReg(spi, Registers.CH7SET.value, conf_CHANNEL)
    writeReg(spi, Registers.CH8SET.value, conf_CHANNEL)
    command(spi, Commands.START.value)
    while(True):
        DRDY_State = GPIO.input(Pins.DRDY_PIN.value)
        while(DRDY_State == GPIO.HIGH):
            DRDY_State = GPIO.input(Pins.DRDY_PIN.value)
        eegReading = readConversion(spi);
        fila.append(eegReading)

if __name__ == '__main__':
    with Manager() as manager:
        d = manager.dict()
        history = manager.list()
        history_graf = manager.list()
        prod = Process(target=productor, args=(history,history_graf))
        prod.start()
        cons = Process(target=consumer, args=(history,))
```

```
        cons.start()
        cons_graf = Process(target=graf, args=(history_graf,))
        cons_graf.start()
        cons_graf.join()
```

B.0.6 Inicializadores - Raspberry Pi.

```
import spidev
import RPi.GPIO as GPIO
import time

PWDN_PIN = 37
RESET_PIN = 35
CLK_PIN = 33
START_PIN = 31
DAISY_IN_PIN = 15
AD_GPIO1_PIN = 13
DRDY_PIN = 3
AD_GPIO4_PIN = 7
AD_GPIO3_PIN = 5
AD_GPIO2_PIN = 11
CS_PIN = 24

def SPI_startup():
  spi = spidev.SpiDev()
  spi.open(0,0)
  spi.max_speed_hz = 1945000
  spi.mode = 0b01   #[CPOL|CPHA],
  spi.cshigh = False
  return spi
  #Raspberry atua apenas com MSBFIRST



def GPIO_startup():
  #Pin initialization
  GPIO.setmode(GPIO.BOARD)
  GPIO.setwarnings(False)
  GPIO.setup(PWDN_PIN, GPIO.OUT)
  GPIO.setup(RESET_PIN, GPIO.OUT)
```

```
  GPIO.setup(CLK_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  GPIO.setup(START_PIN, GPIO.OUT)
  GPIO.setup(DAISY_IN_PIN, GPIO.OUT)
  GPIO.setup(DRDY_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  GPIO.setup(CS_PIN, GPIO.OUT)
  GPIO.setup(AD_GPIO1_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  GPIO.setup(AD_GPIO2_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  GPIO.setup(AD_GPIO3_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  GPIO.setup(AD_GPIO4_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  GPIO.output(PWDN_PIN, GPIO.HIGH)
  GPIO.output(RESET_PIN, GPIO.HIGH)
  GPIO.output(START_PIN, GPIO.LOW)
  GPIO.output(DAISY_IN_PIN, GPIO.LOW)
  GPIO.output(CS_PIN, GPIO.HIGH)


def ADS_hardReset():
  #Hard Reset
  time.sleep(0.15)
  GPIO.output(RESET_PIN, GPIO.LOW)
  time.sleep(0.01)
  GPIO.output(RESET_PIN, GPIO.HIGH)
  time.sleep(0.01)
```

B.0.7 Comandos para o ADS1298 - Raspberry Pi.

```
import spidev

#SYSTEM COMMANDS
WAKEUP=0x02 # Wakeup from standby mode
STANDBY=0x04 # Enter standby mode
RESET=0x06 # Reset the device
START=0x08 # Start/restart (synchronize) conversions
STOP=0xA0 # Stop conversion
#DATA READ COMMANDS
RDATAC=0x10 # Enable Read Data Continuous mode. (default)
SDATAC=0x11 # Stop Read Data Continuously mode
RDATA=0x12 # Read data by command; supports multiple read back.
#REGISTER READ COMMANDS
REG_MASK=0x1F
```

```
RREG=0x20 # Read registers
WREG=0x40 # Write registers

# DEVICE SETTINGS (READ-ONLY REGISTERS)
ID=0x00

# GLOBAL SETTINGS ACROSS CHANNELS
CONFIG1=0x01
CONFIG2=0x02
CONFIG3=0x03
LOFF=0x04

#CHANNEL-SPECIFIC SETTINGS
CH1SET=0x05
CH2SET=0x06
CH3SET=0x07
CH4SET=0x08
CH5SET=0x09
CH6SET=0x0A
CH7SET=0x0B
CH8SET=0x0C
RLD_SENSP=0x0D
RLD_SENSN=0x0E
LOFF_SENSP=0x0F
LOFF_SENSN=0x10
LOFF_FLIP=0x11

#LEAD-OFF STATUS REGISTERS (READ-ONLY REGISTERS)
LOFF_STATP=0x12
LOFF_STATN=0x13

#GPIO AND OTHER REGISTERS
GPIO=0x14
PACE=0x15
RESP=0x16
CONFIG4=0x17
WCT1=0x18
WCT2=0x19
MAX_REG_COUNT=0x1A
```

```
def readReg(spi,reg):
    ret = spi.xfer([RREG | reg,0,0])
    return ret[2];

def writeReg(spi, reg, val):
    print(val)
    ret = spi.xfer([WREG | reg,0,val])
    return;

def command(spi, c):
    ret = spi.xfer([c])

eegLastReading = [0]*9
def readConversion(spi):
   env = [RDATA] + [0]*3*9
   ret = spi.xfer(env)
   for i in range(9):
      num =  ret[3 * i + 1] << 16
      num |= ret[3 * i + 2] << 8
      num |= ret[3 * i + 3] << 0
      if num>0x800000:
         num -= 0x1000000
      num *= 3.3/0x800000
      eegLastReading[i] = num
   return eegLastReading
```