FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Desenvolvimento de um sistema de monitorização e análise em tempo real de actividade neuronal utilizando um DSP

Afonso Maria da Cunha Reis dos Santos e Silva

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: António Pedro Rodrigues Aguiar

Co-orientador: Paulo de Castro Aguiar

29 de julho de 2021

Resumo

A presente dissertação decorre da intersecção entre as áreas da neurociência e da engenharia electrotécnica e de computadores, tendo na sua génese uma parceria estabelecida com o laboratório de Neuro-engenharia Computacional e Neuro-ciência (NCN) do Instituto de Investigação e Inovação em Saúde (i3S).

O estudo elaborado visa o desenvolvimento de um sistema de monitorização e análise em tempo real de redes neuronais, *in vitro*, através de um *digital signal processor*, e de uma *user interface*, que servirá de base ao desenvolvimento de um sistema de controlo em malha fechada destas redes.

O desenvolvimento do código de monitorização foi dividido em duas fases, num primeiro momento começando-se pelo desenvolvimento de um sistema de simulação de dados neuronais e, num segundo momento, procedeu-se à realização do sistema definitivo através da monitorização real de células neuronais.

Podem distinguir-se cinco partes no estudo elaborado. Inicialmente colocou-se particular ênfase na análise do sistema utilizado e do processador nele contido (respectivamente o MEA2100-256 da *MultiChannel Systems* e o processador TMS320C6454 da *Texas Instruments*), destinados à aquisição e processamento dos eventos detectados, nomeadamente *spikes*, *bursts* e *network bursts*. Ainda nesta fase, estudou-se o sistema utilizado para simular os sinais da matriz de micro-eléctrodos e validar os algoritmos desenvolvidos.

Na segunda parte da tese desenvolveu-se os algoritmos de detecção e o processo de validação dos mesmos, quer no sistema de simulação, quer no sistema real, analisando-se também a composição da estrutura geral do código, escrito na linguagem *C*.

A terceira parte do estudo destinou-se ao processo de desenvolvimento da *user interface* na linguagem de programação *C#*.

Em seguida foi desenvolvida a comunicação entre *DSP* e computador.

Por fim, numa última fase desta dissertação, procedeu-se à demonstração de resultados da investigação através da aplicação do sistema desenvolvido à detecção de eventos de actividade neuronal em tempo real. Tendo-se conseguido implementar o sistema proposto, monitorizando pelo menos até 128 eléctrodos em tempo real. Conseguimos também enviar os instantes em que se dão todas estas detecções, visualizá-los e gravá-los em ficheiros de texto para posterior análise. Através da *user interface* desenvolvida conseguimos definir que eléctrodos pretendemos monitorizar e quais os parâmetros requeridos para os algoritmos de detecção. Depois de validado e caracterizado, o sistema desenvolvido foi utilizado para monitorizar diferentes tipos de culturas neuronais.

Palavras-chave — neuro-ciência, células neuronais, actividade neuronal, *digital signal processors*, *MEA2100-256*, tempo real, *spikes*, *bursts*, *network bursts*, matriz de micro-eléctrodos, *user interface*.

Abstract

The present dissertation interfaces the areas of neuroscience and electrical and computers engineering, having in its root a partnership with the Neuroengineering and Computational Neuroscience Laboratory from Instituto de Investigação e Inovação em Saúde (i3S).

The goal of this study is to develop a real-time *in vitro* neural network monitor and analysis system, through a digital signal processor and a user interface, that is going to be the base of the development of closed-loop control system of this networks.

The development of the monitoring code was divided in two phases, in a first moment starting the development of a simulation system of neural data and, in a second moment, procceded the development of the definitive system through the monitoring of real neural cells.

We can distinguish five parts on the elaborated study. Initially particular emphasis was placed on the analysis of the utilized system and processor contained therein (respectively MEA2100-256 from MultiChannel Systems and TMS320C6454 processor from Texas Intruments), destined to the aquisition and processing of the detected events, namely spikes, bursts and network bursts. Still in this phase, we studied the system used to simulate the microelectrode array and validate the developed algorithms.

In the second part of this thesis we developed the detection algorithms and their process of validation, either in the simulation system or in the real system, also analysing the general composition of the structure of the code, writen in *C* language.

The third part of the study was destined to the developing process of the user interface in *C#* programming language.

Then the comunication between *DSP* and computer was developed.

Lastly, in a last phase of this dissertation, we procceded to the demonstration of results of the investigation through the developed system application to neural activity event detection in real time. Having been able to implement the proposed system, monitoring at least 128 electrodes in real-time. We managed to send the instants of every detection, visualize and record them into text files for further analysis. Through the developed user interface the user may define the electrodes he pretends to monitor and the parameters required by the detection algorithms. After its validation and description, the developed system was used to monitor different types of neural cells.

Keywords — neuroscience, neural cells, neural activity, digital signal processors, *MEA2100-256*, real time, spikes, bursts, network bursts, microelectrode array, user interface.

Agradecimentos

Gostaria de agradecer ao meu orientador e co-orientador, Professor Doutor Pedro Aguiar e Professor Doutor Paulo Aguiar por toda a ajuda e orientação oferecida no decorrer deste trabalho. Quero agradecer em especial ao Domingos Leite Castro, que acompanhou de perto todo o desenrolar desta tese e foi uma ajuda preciosa tanto no desenvolvimento como na escrita da dissertação.

Por fim, agradeço aos meus familiares por me darem todas as ferramentas e apoio necessários para o meu desenvolvimento enquanto pessoa e estudante.

Afonso Maria da Cunha Reis dos Santos e Silva

"O segredo da existência humana reside não só em viver
mas também em saber para que se vive."

Fiódor Dostoiévski, Os irmãos Karamázov

Conteúdo

Lista de Figuras

Abreviaturas e Símbolos

BMI	Brain-Machine Interface
CCS	Code Composer Studio
CPU	Central Process Unit
DLL	Dinamic Link Library
DMA	Direct Memory Access
DRG	Dorsal Root Ganglion
DSP	Digital Signal Processing
FIFO	First In First Out
EEG	Electroencephalography
IDE	Integrated Development Environment
LCDK	Low Cost Development Kit
MCS	MultiChannel Systems
MEA	Micro Electrode Array
NCN	Neuro-engenharia Computacional e Neuro-ciência
TI	Texas Instruments
UART	Universal Asychronous Receiver/Transmitter

Capítulo 1

Introdução

A neurociência engloba uma variedade de questões acerca de como o sistema nervoso humano e de outros animais está organizado, como se desenvolve e como funciona para gerar comportamentos [1]. Já desde o século XIX, através de Ramon y Cajal e Golgi, que os cientistas revelam interesse no estudo do sistema nervoso. Vários estudos foram sendo realizados de maneira a termos um conhecimento amplo acerca do funcionamento do sistema nervoso central e periférico. Estes estudos oferecem-nos informação acerca do funcionamento das principais células do nosso sistema nervoso, neurónios e glia, e que zonas do nosso cérebro processam o quê. Dizem-nos que este está compartimentado em zonas de processamento definidas para diferentes tarefas, sejam estas de aprendizagem, processamento visual, sonoro, etc. Também nos dão conhecimento de como funciona a transmissão de informação entre neurónios, eléctrica ou bioquímica (neurotransmissores), que influência têm estes neurotransmissores no tipo de informação a ser transmitida, e também como o sistema nervoso pode ser influenciado pela nossa carga genética.

Ao longo dos últimos dois séculos surgiram enúmeras novas formas de analisar estes sistemas e também diferentes áreas de estudo de modo a podermos compreender melhor o seu funcionamento, com o objectivo final de podermos detectar e suprimir (através de fármacos, ou outro tipo de estratégias) certas doenças ou anomalias associadas ao sistema neuronal. Uma dessas estratégias de análise consiste na interface entre dispositivos electrónicos e culturas de redes de células neuronais *in vitro*, onde são realizados estudos em técnicas de gravação de actividade e estimulação de neurónios.

Com base na estratégia referida, nesta tese pretende-se desenvolver e explorar um sistema usando um *Digital Signal Processor* (*DSP*), que permite monitorizar e calcular métricas de actividade neuronal em tempo real. O *DSP* está integrado num sistema de electrofisiologia que combina informação de 252 eléctrodos. No contexto desta tese são usadas culturas neuronais *in vitro*, mas os princípios deste trabalho estendem-se a experiências *in vivo* (p.e. sistemas de matrizes eléctrodos implantáveis em humanos, ou brain-machine interfaces).

1.1 Motivação

A principal motivação para o desenvolvimento deste tipo de sistemas é, sem dúvida, a cura (idealmente) ou tratamento de doenças de cariz de degenerativo (Parkinson, Alzheimer, etc.), problemas mentais, cada vez mais presentes nos dias que correm, tais como a depressão [2], esquizofrenia [3] ou perturbações bipolares [4], problemas motores (p.e. paraplegia) [5], epilesia, etc.

Como sabemos já existem tratamentos para estas doenças a nível de medicamentação ou mesmo através do uso de *ECT* (terapia de electro-estimulação), que voltou a ser utilizada, em casos mais graves, onde outros tratamentos não sortiram efeito [6]. No entanto, todos estes tratamentos afectam muitas outras áreas do nosso corpo, podendo ter efeitos indesejáveis [7][8][9]. Mais, o progresso na descoberta de novos medicamentos para doenças como Parkinson [10], Alzheimer [11] e epilepsia [12] tem sido consideravelmente lento. Uma vez que a função neuronal está intimamente associada à actividade electrofisiológica, tem sido dada atenção a um tipo diferente de abordagem: modulação directa da actividade neuronal através de estimulação eléctrica [13]. Estimulação cerebral profunda para doença de Parkinson [14], estimulação da medula espinal para dor crónica ou implantes cocleares [15] para perda auditiva são alguns exemplos de sucesso que demonstram o potencial desta estratégia. A utilização de eléctrodos implantados em áreas específicas do cérebro, associados a um sistema inteligente de processamento de sinal e activação de estimulação eléctrica, poderão, um dia, mitigar estas doenças a um nível mais localizado e controlado.

Para além destas motivações, que por si só já são amplas, existe também a motivação de obter um maior conhecimento do funcionamento do nosso sistema nervoso, que poderá trazer-nos respostas sobre os nossos comportamentos, a nossa forma de pensar, aprender, processar, adquirir memórias, etc.

1.2 Contexto

Esta tese está enquadrada no trabalho de investigação realizado pelo grupo *Neuroengineering and Computational Neuroscience (NCN)*, do Instituto de Investigação e Inovação em Saúde (i3S). O *NCN* desenvolve e utiliza interfaces neuro-electrónicas para analisar e controlo de redes neuronais. Utilizam o sistema *MEA2100-256-System (Multichannel Systems, MCS)* para fazer aquisição e estimulação de actividade neuronal.

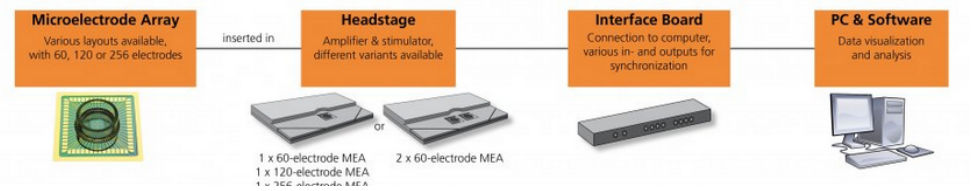

Figura 1.1: Diagrama geral do sistema [16]

Esta proposta de tese tem como origem a necessidade de adquirirmos melhores capacidades de processamento em tempo real de informação de uma rede neuronal, *in-vitro*, através de *micro electrode arrays (MEAs)*, de modo a conseguirmos no futuro fazer o controlo em malha fechada. Precisamos de conceber um sistema que seja capaz de analisar a actividade capturada em múltiplos eléctrodos em tempo real para detectar métricas/eventos neuronais de interesse. Para isso, será usado o *DSP* que se encontra na *Interface board*, na qual os dados são recebidos, vindos do *headstage*, através de um cabo *eSata*.

Análise e actuação em tempo real é essencial para se poder fazer o controlo em malha fechada, ou mesmo analisar uma rede em tempo real. Normalmente, o sinal é analisado pelo computador, o que leva a uma impredictibilidade de atrasos do sinal de pelo menos 100 ms. Ao fazer esta análise num *DSP* em vez de num computador o atraso é reduzido para menos de 1 ms.

Esta tese centrar-se-á em explorar as capacidades de classes específicas de *DSPs* na monitorização em "tempo real"da actividade das populações neuronais, com foco nos *TMS320C6454*, pois é disponibilizado em sistemas de electrofisiologia (como o *MEA2100* da *Multichannel Systems*). Sendo que o trabalho realizado no final desta dissertação será utilizado como base para a realização de um sistema que seja capaz de controlar as dinâmicas de actividade eléctrica de redes de neurónios em tempo real.

1.3 Objetivos

Este trabalho irá focar-se no uso de um *DSP* que receberá a informação já digitalizada do *MEA* para monitorizar os principais eventos da actividade neuronal em tempo real:

1. Processamento em tempo real da informação neuronal
   (a) Implementação de um filtro passa alto para monitorização de células
   (b) detecção de eventos:
      i. *neural spikes*
      ii. *neural bursts*
      iii. *network bursts*
   (c) métricas de atividade:
      i. *inter-spike interval*
      ii. *burst duration*
      iii. *burst size (number of spikes in burst)*

2. Visualização e análise preliminar de resultados em "tempo real":
   (a) Exportar os eventos de actividade detectados para o PC com uma periodicidade regular

(b) Desenvolver interface para monitorizar a actividade exportada em tempo real

3. Caracterização da performance do *DSP*

1.4 Estrutura da Dissertação

Para além desta introdução, a presente dissertação foi dividida em 7 capítulos diferentes. No capítulo 2, é feita uma revisão de conceitos na área de neurociência, em seguida fazemos a descrição do estado da arte e são apresentados alguns trabalhos relacionados. No capítulo 3 é feita a descrição do sistema de monitorização utilizado no laboratório e também do sistema que foi desenvolvido para o simular. No capítulo 4 demonstra-se como foram pensados e validados os algoritmos de detecção, bem como as estruturas de dados utilizadas. No capítulo 5 mostra-se como foi desenhada a interface de utilizador. No capítulo 6 fazemos a análise da elaboração do sistema de comunicação entre *DSP* e *PC*. No capítulo 7 validamos o sistema desenvolvido, analisamos o filtro passa alto utilizado para fazer a detecção de eventos dos sinais neuronais e em seguida mostramos os dados recolhidos de leituras feitas com este tipo de células por parte do sistema desenvolvido. No capítulo 8 concluímos com uma análise geral de todo o processo realizado para desenvolver o sistema de monitorização em tempo real e fazemos referência ao trabalho que pode ser feito no futuro para melhorar o sistema e para iniciar o controlo em malha fechada.

Capítulo 2

Revisão Bibliográfica

Esta secção apresenta a análise de alguma da literatura existente referente ao processamento digital de sinal da actividade eléctrica de células neuronais. Começa-se por definir como se comportam este tipo de células a nível da sua actividade eléctrica, depois explora-se a forma como pode ser feita a aquisição destes dados de maneira a retirarmos informação que nos possa ser útil para o seu estudo.

2.1 Actividade neuronal

O cérebro é perito em adquirir, coordenar e disseminar informação acerca do corpo e do seu ambiente, sendo esta informação processada em fracções de segundo de modo a haver uma resposta aos estímulos o mais rápida possível. Os neurónios do sistema nervoso central e periférico desempenham estas funções através da geração de sinais eléctricos e químicos. O local onde ocorrem estes fenómenos de troca de informação entre neurónios chama-se sinapse, onde um potencial de acção despoleta a secreção, para a fenda sináptica, de neurotransmissores [17]. Normalmente, a actividade cerebral é resultante de sinapses químicas, no entanto os neurónios podem ser excitados por um campo eléctrico [18], existindo aplicações clínicas que utilizam este princípio [19]. Portanto, é importante a investigação nesta área de modo a conhecermos os efeitos que esse tipo de estimulação possa ter nas células neuronais. Vamos focar-nos então apenas na actividade eléctrica deste tipo de células, já que esta é uma das bases fundamentais dos neurónios[20][21].

As células nervosas, não sendo naturalmente boas condutoras de electricidade, desenvolveram mecanismos para gerar estes sinais baseados no fluxo de iões através das suas membranas celulares. Normalmente, os neurónios encontram-se com uma polarização negativa das suas membranas, chamado potencial de membrana em repouso, entre o interior e o exterior da célula [22]. Os neurotransmissores quando entram em contacto com a membrana neuronal do receptor alteram o potencial da membrana que estava previamente em repouso. Se este potencial tiver um valor maior que um certo limite, então é gerado um potencial de acção que irá despoletar uma nova descarga de neurotransmissores. Estes potenciais de acção, *spikes*, são propagados ao longo dos axónios e levam assim a informação a todo o tipos de sistemas do nosso corpo. Há ainda outros tipos de

sinais eléctricos que são produzidos pela activação de contactos sinápticos entre neurónios ou pela acção de formas de energia externa, como som, luz, ou contacto.

Os neurónios utilizam vários tipos de sinais eléctricos para codificar e transferir informação [23], a forma mais eficaz de observar estes sinais é através do uso de micro-eléctrodos intracelulares. Podemos então observar, inicialmente, o potencial negativo constante que há pouco falávamos, que ronda entre os -60 e os -70 mV [1]. Na figura 2.1 podemos observar os diferentes estados do potencial de um neurónio.

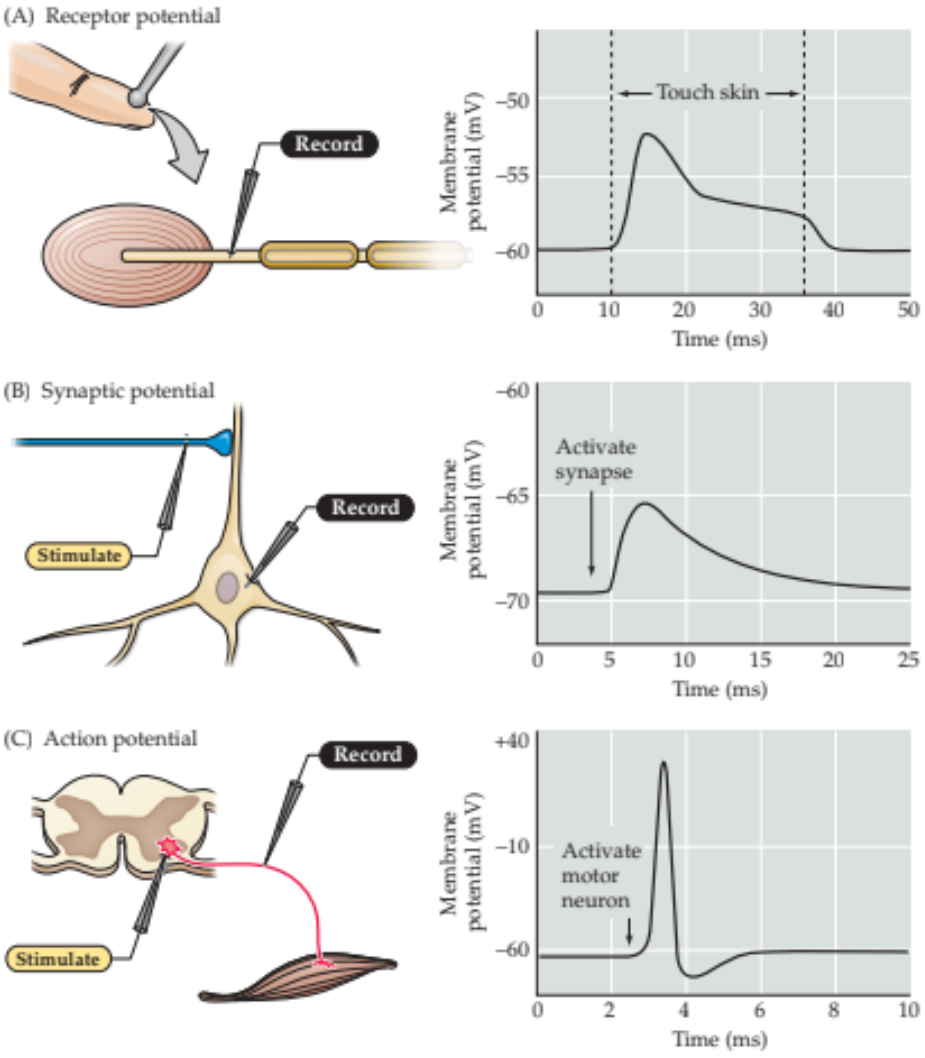

Figura 2.1: Potencial eléctrico de um neurónio [1]

Na figura 2.2 podemos ver as diferentes respostas que vários tipos de estímulos geram. Podemos produzir um potencial de acção fazendo passar corrente através da membrana do neurónio, mas no caso de esta diminuir o potencial da célula nada acontece, sendo assim uma resposta passiva. Por outro lado, no caso da corrente administrada ter uma polaridade contrária, se esta au-

mentar o potencial da membrana acima de um determinado valor, *threshold*, o potencial de acção ou *spike* ocorre.

O potencial de acção é a resposta gerada pelo neurónio, é tipicamente breve, cerca de 1 ms, e é independente da magnitude da corrente utilizada para o invocar. Em vez disso, se a amplitude ou duração do estímulo for aumentada, múltiplos *spikes* ocorrem.

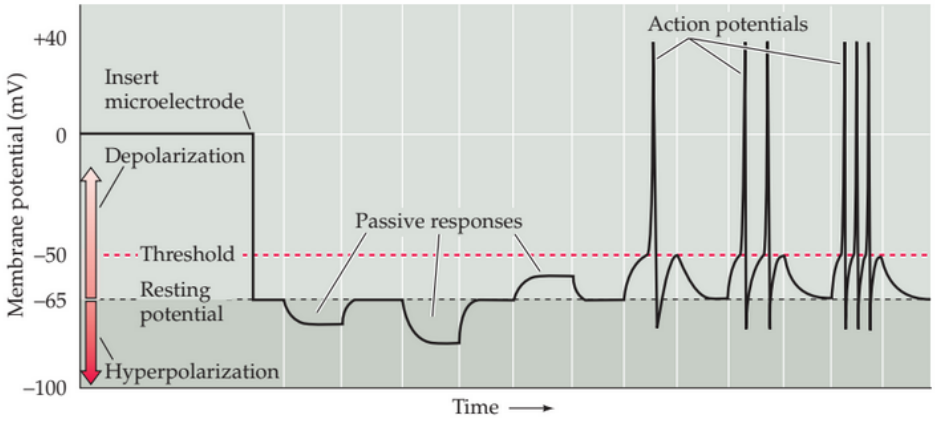

Figura 2.2: Potencial de acção [1]

2.1.1 *Microelectrode Array*

A tecnologia *MEA* é extremamente utilizada no estudo de redes neuronais *in vitro*, já que permite gravar informação, a longo prazo, ou estimular neurónios, acedendo a várias zonas da mesma rede em simultâneo [24]. Esta técnica é designada, ao contrário da analisada anteriormente, extra-celular, já que não estamos a aceder ao interior de cada neurónio para analisarmos o seu comportamento eléctrico, mas sim a receber essa informação através do contacto de um eléctrodo com a parte exterior de um conjunto de células (figura 2.3). Ao contrário das técnicas intra-celulares, esta tecnologia oferece uma interface não invasiva, podendo assim fornecer informação durante longos períodos de tempo[25][26].

Os *MEAs* foram criados em 1972, e já foram utilizados desde então em vários estudos nos campos de engenharia neuronal. A cultura neuronal neste sistema pode manter actividade durante cerca de um mês, se forem controlados a temperatura e o *pH* em níveis aceitáveis.

Existem diferentes tipos de sistemas *MEA*, nomeadamente, sistemas com até 256 micro-eléctrodos, num substrato plano (*PCB*, ou vidro), ou sistemas *CMOS* com milhares de eléctrodos. Os neurónios são colocados na superfície do *MEA* e entram em contacto directo com os eléctrodos. Para colocar os neurónios, ou cultivá-los é necessário um anel de vidro para suster a cultura.

Os neurónios cultivados nos *MEAs* apresentam actividade espontânea e cooperativa, a rede tende a apresentar eventos em sintonia, que são normalmente chamados de *network bursts*. Nas seguintes figuras podemos observar como se comporta a rede neuronal, bem como o tipo de dados que vão ter que ser recolhidos. Na figura 2.4 a imagem da esquerda apresenta a sequência temporal

de tensão, amplificada, captada por cada eléctrodo, já a figura da direita mostra-nos apenas em que momentos, desse mesmo intervalo de tempo, foi captado um *spike*, que é uma forma bastante mais compacta de gravar a informação. Isto é também importante porque a informação mais relevante da *linguagem neuronal* é saber se existe ou não um *spike*.

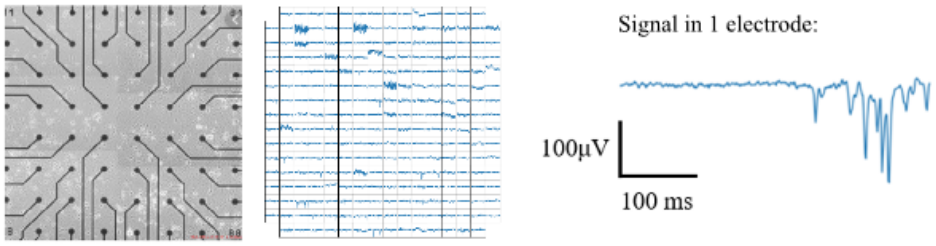

Figura 2.3: Aquisição de dados num dado intervalo de tempo

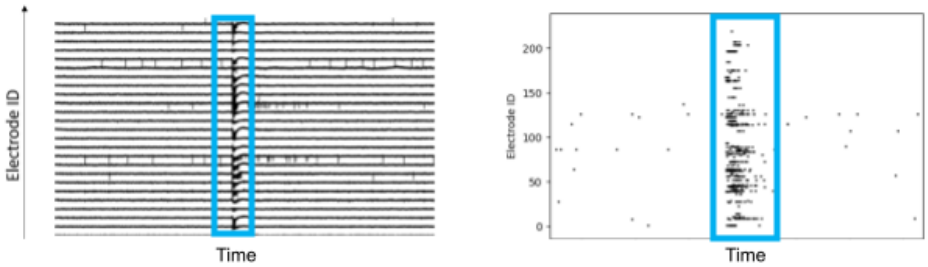

Figura 2.4: Actividade Neuronal

2.1.2 Estado da Arte

Existem vários modos de fazermos estimulação neuronal, através de correntes ou campos eléctricos [18] como, por exemplo, através de aplicação de campos eléctricos em certos eléctrodos pertencentes a um *MEA*, campos electromagnéticos (p.e. luz, ou produção de campos electromagnéticos de baixa frequência [27]), químicos, etc.

O uso de culturas neuronais de ratos dissociadas é muito comum nesta área da neuro-ciência. Depois de alguns dias *in vitro* estas redes apresentam alta actividade, com comportamentos análogos à actividade neuronal humana [28].

A estimulação eléctrica é usada no estudo de células neuronais e redes constituídas por elas, pode ser feita *in vivo*, *in vitro*, ou *in silico* [29]. Ou seja, num ser vivo, numa amostra, ou numa simulação computacional. No estudo destas células *in vitro*, que é a forma utilizada pelo nosso sistema, são usadas fatias de cérebro, ou culturas de células neuronais. O desenvolvimento de técnicas de controlo complexas directamente *in vivo* é bastante desafiante e pouco seguro, portanto o uso de ambientes *in vitro* são bastante importantes para a testagem destes sistemas de controlo.

Actualmente, esta técnica é feita em *open loop*, ou seja, aplicam-se estímulos e observam-se as respostas. Os cientistas estão neste momento a tentar recriar redes controladas em *closed loop* [30][31], o que poderá trazer muito mais conhecimento acerca destas, ajudando a perceber o funcionamento do nosso cérebro, de modo a que um dia seja possível curar doenças por este meio. Neuro estimulação em malha fechada poderá vir a ser um tratamento bastante eficaz para diversas doenças, no entanto, até agora os resultados mostraram que a estimulação em malha fechada e malha aberta (mais simples) apresentam eficácia semelhante. Neste momento usam-se estes sistemas para fazer análises estatísticas de eventos na rede neuronal, quer seja com ou sem estimulação eléctrica. Sendo que a estimulação é criada através de uma sequência de impulsos na casa das centenas de milivolts[32]. Já é possível através de *DSPs* por exemplo, fazer o processamento destes dados em tempo real, e tendo estes processadores elevada capacidade computacional, são uma das tecnologias mais indicadas para este tipo de experiência.

2.1.3 DSPs no contexto da Neurociência

Existem algumas publicações científicas que explicitam o método como realizam as experiências em tempo real através de *DSPs* [18][33][34].

O processo pode começar pela utilização prévia de um filtro passa alto de resposta impulsional infinita, através de um *FPGA*, ou pode ser feito posteriormente à digitalização e aquisição dos dados pelo processador por um filtro de resposta impulsional finita. O caso em que filtragem é feita anteriormente à aquisição dos dados pelo *DSP* é mais eficiente, pois não há tempo de processamento desperdiçado na filtragem, por parte do *DSP*. Esta filtragem permite que se reduza o nível de ruído e que seja eliminado o *offset* de baixa frequência.

No que toca à aquisição dos dados pelo DSP, é normalmente utilizado o *Direct Memory Access (DMA)*, que permite o acesso direto à memória do DSP independentemente do CPU [16][35]. Após ter sido feita a aquisição e filtragens dos dados é necessário estabelecer um valor de *threshold*, para os valores medidos em cada amostra, de modo a ser feito o algoritmo de *spike detection*.

Nas experiências [18] em que é implementada estimulação eléctrica são usadas amostras de células que estão sujeitas a estímulos e outras que não têm qualquer estímulo, exibindo actividade espontânea, podendo assim ser feita a comparação de resultados.

Existem também experiências nas quais são usados algoritmos, num *DSP*, capaz de reproduzir o comportamento de uma rede neuronal real, chamado *Spiking Neural Networks*. Estes algoritmos de redes neuronais artificiais poderão a ser muito importantes no controlo em malha fechada de sistemas reais, pois seremos capazes de prever o resultado de um estímulo numa rede neuronal, para além de podermos treinar a rede artificial para se comportar como a rede neuronal em causa. Neste caso, existiria uma interacção em tempo real entre a rede neuronal *in vitro* e a rede neuronal *in silico* [36].

Foram também implementados sistemas robóticos que usam redes neuronais *in vitro* para controlar um robot de modo a que este evite obstáculos [37]. No entanto, até agora estas experiências

foram apenas demonstrações sem grande utilidade real e o seu valor científico ainda é bastante limitado.

2.1.4 Brain-machine interfaces(BMI)

Já desde finais do século passado que cientistas estudam formas de conectar áreas do cérebro a dispositivos externos, como computadores. Estes estudos podem ser direccionados para diversas áreas sejam elas para mapeamento do cérebro, monitorização, reparação cognitiva e de capacidades motoras, etc [38].

Existem dois métodos utilizados para monitorização da actividade neuronal para estes sistemas, invasivos (intra-cranianos) e não invasivos. Os sistemas não invasivos usam principalmente electroencefalogramas (*EEGs*), que têm demonstrado ser uma boa solução para pacientes paralisados, de modo a que eles possam comunicar com o exterior, não os expondo ao risco de cirurgia cerebral [39]. Não sendo esta a melhor opção no que toca ao controlo de membros em tempo real. Já os sistemas invasivos, implantados directamente na matéria cinzenta do cérebro, baseiam-se na monitorização da actividade de conjuntos de neurónios [40], sendo estes os que captam sinais de maior qualidade. No entanto, podem surgir problemas se o corpo rejeitar tais implantes.

Um dos principais focos dos dispositivos baseados em *BMI*, neste momento, é a restituição de capacidades motoras em pacientes que sofrem de doenças neurológicas, perda de membros, danos cerebrais ou visuais [41][42]. Para tal, têm sido levado a cabo estudos feitos em roedores [43] e primatas [44]. Nestes estudos são implantados *MEAs* em várias áreas corticais do cérebro dos animais, onde um braço robótico é controlado através da actividade cortical que controla os movimentos, sendo descodificada através de algoritmos[1] que transformam inúmeros *spikes* em sinais contínuos que controlam o movimento do braço robótico.

2.2 Conclusões

Podemos concluir que se conseguirmos monitorizar o comportamento destas células, em tempo real, através dos seus impulsos eléctricos, identificar quando é que estas redes entram em comportamentos patológicos como, por exemplo, exacerbada sincronia que ocorre em doenças como *Parkison* e epilepsia. A detecção destes fenómenos e intervenção em tempo real poderá contribuir para a manutenção de uma rede funcional. Uma vez que esta compreensão esteja cimentada e possamos monitorizar estes impulsos em tempo real, a longo prazo, podemos também controlá-los e minimizar os efeitos destes problemas.

À medida que fizemos esta revisão de literatura, apesar do entusiasmo e também dos muitos avanços feitos, apercebemo-nos que há ainda muitas experiências e estudos a serem feitos antes que os *BMIs* sejam seguros e eficientes o suficiente para que possam ser utilizados na reabilitação de pacientes. Algumas das coisas que têm que ser melhoradas são a qualidade da monitorização neuronal, alcançar performances estáveis para períodos de tempo longos, tornar as cirurgias para

[1] Estes algoritmos correm em múltiplos computadores em paralelo de modo a garantirem um controlo em tempo real

implantação destes dispositivos no cérebro mais seguras, perceber qual é o efeito destes implantes a longo prazo, etc.

Por fim, verificamos que existem muitos estudos e trabalhos publicados no que diz respeito a monitorização de actividade eléctrica para controlo de sistemas externos, p.e. braços robóticos, ou ou aquisição de informação externa para estimular a actividade eléctrica. No entanto, isto já não se verifica para controlo da própria actividade que está a ser monitorizada, que será o objectivo do sistema que desenvolvemos. Sendo que podem ser realizados algoritmos para fazermos controlo em malha fechada.

Capítulo 3

Sistema de Aquisição e Processamento de Sinais

Este capítulo é dedicado à descrição do sistema de aquisição e processamento de sinais neuronais e está dividido em cinco secções. Na secção 3.1 apresenta-se a análise do funcionamento do sistema *MEA2100-256 System (MCS)* de aquisição e processamento de sinais neuronais, presente no laboratório, e algumas ferramentas de software que a *Multichannel Systems* providencia e que serão bastante importantes na fase de validações. A descrição do *DSP* nele integrado está presente na secção 3.1.3 e conceito de sistema encontra-se na secção 3.3. Uma vez que o acesso ao sistema *MEA2100* (e o respectivo *DSP*) é bastante limitado, grande parte do código foi desenvolvido e validado utilizando um kit extra, *Low-Cost Development Kit (LCDK)*, que inclui um *DSP* semelhante ao do sistema. A secção 3.2.1 descreve o funcionamento do *LCDK* utilizado e a secção 3.4 é dedicada a descrever as ferramentas que serão necessárias para o desenvolvimento de *software*.

3.1 MEA2100-256

O *MEA2100-256* é um sistema compacto e portável para medição de sinais eléctricos extracelulares através de *MEAs* com amplificação integrada, aquisição de dados, processamento em tempo real, gerador de estímulos e controlo de temperatura[45]. Todo o sistema é controlado através de um *PC* com software específico para o efeito e utilizado também para visualização de dados e análise.

O nosso sistema contém 256 canais de eléctrodos, sendo que quatro deles funcionam como massas do sistema. É capaz de fazer amostragem até a uma frequência de 50 kHz por canal. O *MEA2100-256 system* é constituído por uma *interface board* e uma *headstage*[16].

Os dados neuronais são capturados através do *Micro-Electrode Array*, que se encontra na *headstage*, no qual pode ser feita uma cultura neuronal ou colocada uma fatia de cérebro.

A *headstage* contém:

- Adaptadores para diferentes *MEAs*

- Amplificador
- Electrónica para medição
- Dois geradores de estímulos independentes
- Conversor *ADC* de 24 bits de resolução

A *headstage* (figura 3.1) é o elemento principal do sistema, já que alberga e aquece o *MEA*, amplifica e digitaliza o sinal. O amplificador faz com que os sinais sejam ampliados próximos da fonte, de modo a minimizar o ruído[1].

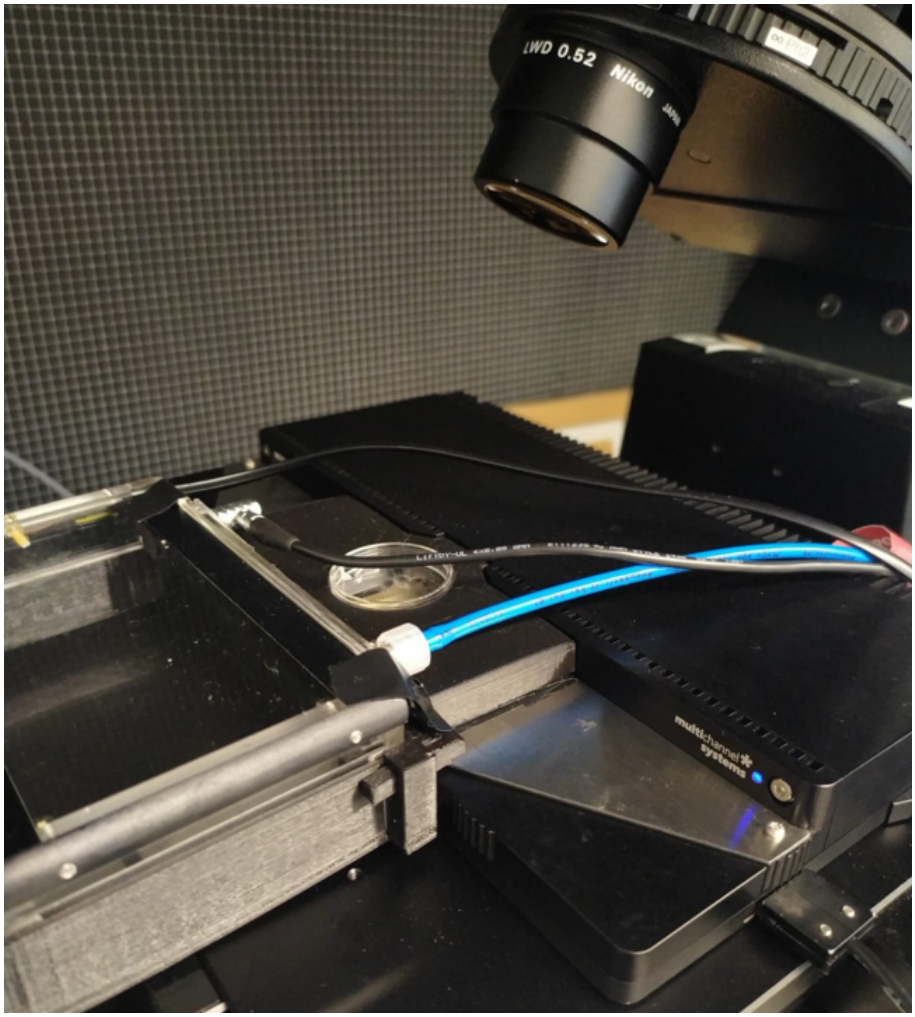

Figura 3.1: *Headstage* da *MCS* usada no laboratório

A *interface board* contém:

- Dois conectores *USB* para comunicação com o computador

[1] https://www.multichannelsystems.com/products/mea2100-systemsdescription

- Duas portas para uma ou duas *headstages*
- 32 portas *I/O* para criar *triggers*
- Oito *inputs* analógicos para monitorização
- Porta de *debug* de *DSP*
- *audio output*
- *FPGA* (apenas podemos alterar alguns registos deste)
- *DSP*

A *interface board* (figura 3.2) recebe os dados provenientes da *headstage* através de um cabo eSata. Aqui podemos encontrar o *DSP*, que iremos usar para detecção em tempo real. A *interface board* é conectada ao computador através de cabo *USB 3.0 SuperSpeed*.

Figura 3.2: *Interfaceboard* da *MCS* usada no laboratório

A figura 3.3 representa a montagem *standard* do sistema, na qual o *DSP* não se encontra a executar nenhuma tarefa, todo o processamento é feito através do computador.

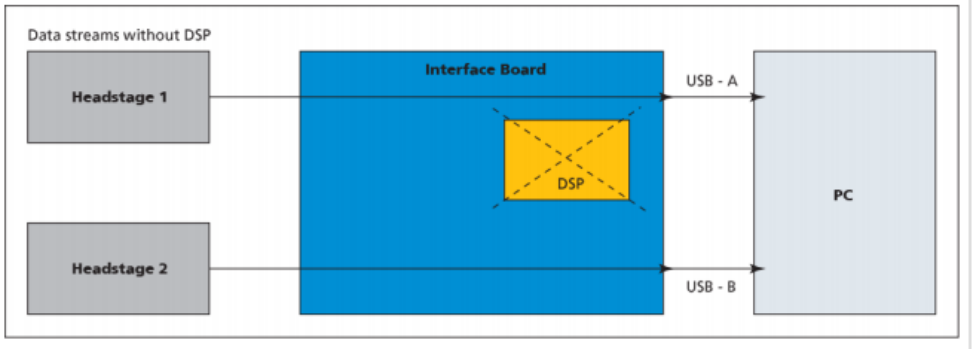

Figura 3.3: Montagem default [16]

Já na figura 3.4 está representada a montagem no qual se faz o processamento em tempo real através do *DSP*. Os dados podem também ser transmitidos em paralelo para o computador, de modo a podermos fazer uma análise posterior dos resultados, ou ainda do *DSP* para o computador.

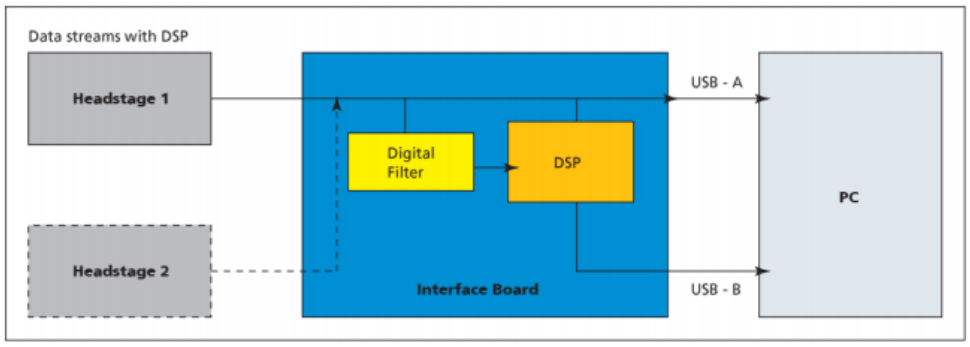

Figura 3.4: Montagem com DSP [16]

3.1.1 Matriz de Eléctrodos

O *MEA* que utilizamos tem 256 eléctrodos, sendo que apenas 252 são utilizáveis para fazer medições, os restantes 4 são utilizados como massas, como podemos observar na figura 3.5. A matriz está organizada por um código alfanumérico, na qual cada cada eléctrodo está identificado por uma letra e um número. A coluna a que pertence um eléctrodo é identificada por uma letra e a linha por um número. À primeira coluna dá-se a letra "A", à segunda a "B"e assim sucessivamente até chegarmos à coluna "R", que é a décima sexta. Do mesmo modo à primeira linha corresponde o número "1", à segunda o "2"e assim sucessivamente.

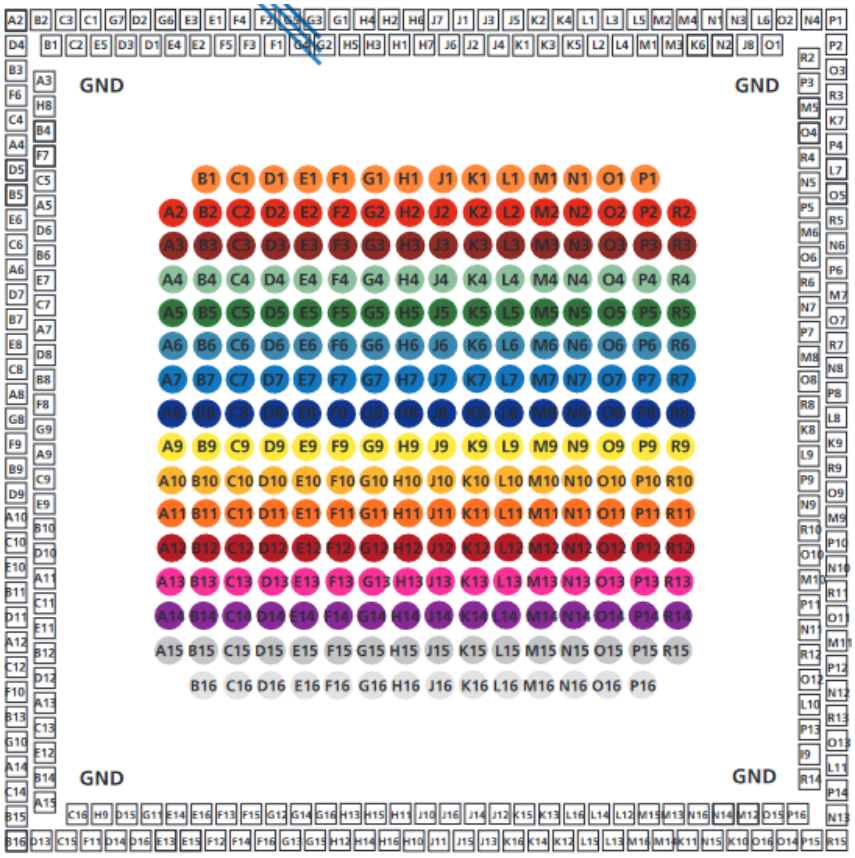

Figura 3.5: Mapeamento de eléctrodos do *MEA*[46]

Existe um pormenor importante referir e que pode gerar alguma confusão, que está ligado ao facto da recepção da informação de cada eléctrodo por parte do *DSP* não estar ordenada da mesma forma que esta matriz. A informação recebida pelo sistema de aquisição num dado vector pode ser por exemplo:

Elec	Vec Pos
G13	6
G12	7
F16	8
F15	9
F14	10
⋱	⋱
⋱	⋱
⋱	⋱
G16	254
H12	255
G14	256

3.1.2 Multi Channel Suite

A *Multichannel systems* disponibiliza dois *softwares*, *Multi Channel Suite* ou *MC_Rack program*, capazes de gravar, estimular e analisar os dados recolhidos. O *software* utilizado no laboratório para análise de dados é o *Multi Channel Suite*, pelo que iremos agora falar deste em mais detalhe.

O *Multi Channel Suite* contém três ferramentas:

- *Multi Channel Experimenter* para aquisição de dados e análise *online* (no *PC*)
- *Multi Channel Analyser* para análise das gravações *offline*
- *Multi Channel DataManager* para exportação de dados para outros programas

Este software oferece uma interface intuitiva para o utilizador fazer análise dos dados recolhidos, como podemos ver na figura 3.6. No entanto, o modo como esta comunicação é feita entre o computador e o *MEA2100* gera uma certa latência inerente à aquisição e análise de dados, já que o computador apenas faz a detecção de eventos após ter feito uma leitura de um período de tempo definido para cada eléctrodo, o que se torna um problema quando queremos gerar impulsos para o controlo dos neurónios em malha fechada. Daí a necessidade de utilizarmos o *DSP* para a aquisição e processamento dos dados adquiridos e eventual controlo da estimulação eléctrica. Existe ainda a possibilidade de usarmos este sistema com o *DSP* a processar paralelamente.

O *Multi Channel Experimenter* servirá também de apoio para validação de algoritmos no *DSP*, pois vai permitir-nos comparar os dados de ambas as partes e ainda controlar o estimulador permitindo-nos criar *spikes* artificiais de forma controladas, como veremos no capítulo 4.

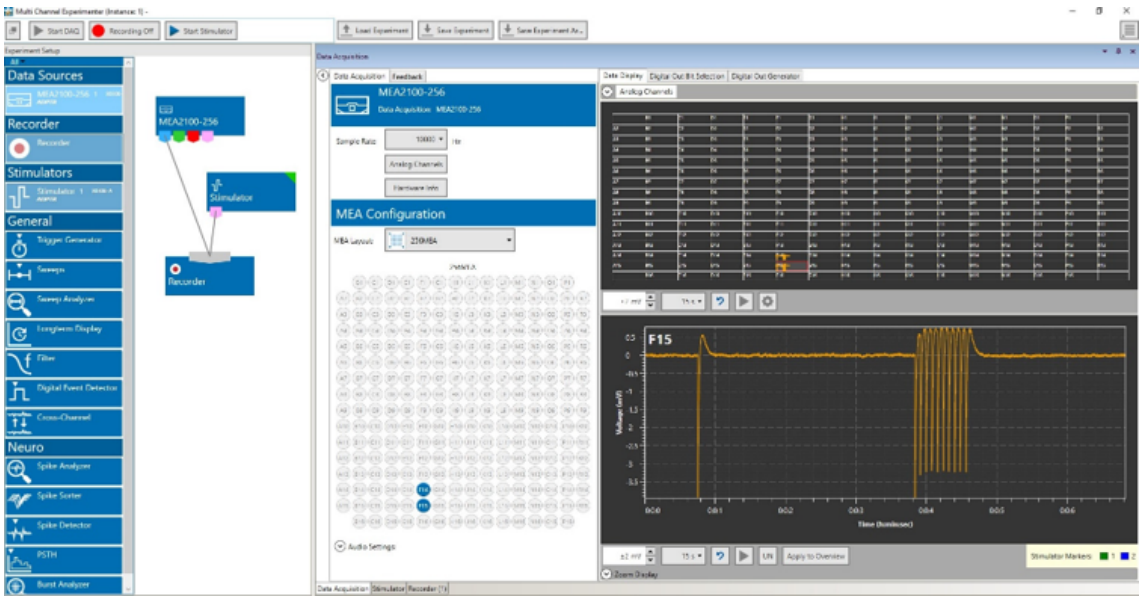

Figura 3.6: *Multi Channel Experimenter layout*

3.1.3 *DSP*

Nesta secção apresentamos o *DSP* utilizado no *MEA2100* para fazer o processamento do sinal dos eléctrodos.

3.1.3.1 C6000

A família de processadores *C6000* da *Texas Instruments* é utilizada para o processamento de alta performance de sinais digitais, incluindo computações matemáticas em tempo real de conjuntos de dados paralelos. Os *TMS320C6000*™ da *TI* utilizam uma arquitectura de *VLIW (very long instrution list)* e apresenta um baixo consumo de potência. A família apresenta processadores com *fixed point*, *floating point*, ou ainda ambos juntos [47].

3.1.3.2 DSP presente no MEA2100-TMS320C6454

O *DSP C6454* é o que se encontra no nosso sistema de aquisição, na *interface board*, será a partir deste que faremos a aquisição e processamento de dados.

O *DSP C6454* funciona a 819.2 MHz. Contém oito unidades funcionais, dois conjuntos de registos e dois caminhos de dados. Duas dessa unidades (.M) funcionais são multiplicadores (que podem realizar mais do que duas operações deste tipo em simultâneo, dependendo do tamanho dos dados, p.e. oito operações de 16bit*16bit), quatro (.S e .L) servem para funções aritméticas e lógica e duas (.D) para fazerem *load* de dados e guardar resultados dos registos para a memória[48].

A montagem *default* do *MEA2100* conecta a *headstage* à porta *USB A* que por sua vez conecta ao computador, sem interacção do *DSP*. No entanto, o *DSP* pode ser programado para aceder ao *stream* de dados ou ao *ADC* da *interface board*.

O DSP tem acesso ao *MEA2100 system* através de um *EMIF (external memory interface)*. Os diferentes componentes do *MEA2100* usam registos que contêm três regiões de endereços mapeadas no *DSP* que podemos encontrar nos manuais do sistema[49] de modo a podermos configurar, aceder aos dados e também criar estímulos:

- A configuração do *MEA2100* tem os endereços mapeados entre 0xA0000000 e 0xAFFFFFFF.
- Os dados são transferidos através de um sistema *FIFO* mapeados na região de endereços começada em 0xB0000000.
- Podemos fazer o controlo de estimulação através dos endereços da região 0xC0000000.

3.1.3.3 Registos da *Interface board*

Todos os registos do *FPGA* presente na *interface board* estão mapeados na região de memória de 0xA0000000 a 0xA000FFFF do *DSP*.

Estes registos são utilizados para realizar todas as configurações da *interface board* através do *DSP*, por exemplo definir quantas *headstages* serão utilizadas, qual será a estrutura das *frames* de dados que o processador recebe, qual será a velocidade de aquisição dos sinais neuronais, etc.

```
0x0000 to 0x0FFF: Interfaceboard
0x1000 to 0x1FFF: Mailbox Registers on Interfaceboard
0x2000 to 0x2FFF: RAM Registers on Interfaceboard
0x8000 to 0x8FFF: Headstage 0
0x9000 to 0x9FFF: STG1 on HS 0
0xA000 to 0xAFFF: STG2 on HS 0
0xC000 to 0xCFFF: Headstage 1
0xD000 to 0xDFFF: STG1 on HS 1
0xE000 to 0xEFFF: STG2 on HS 1
```

Figura 3.7: Mapeamento de registos[16]

3.1.4 Aquisição de dados

Os dados são transferidos da *headstage* para a *interface board* através de um cabo *eSATA*.

Existem duas formas de fazer a aquisição de dados. A primeira consiste numa recepção de um sinal de *trigger* pelo *DSP* informando que existe um novo conjunto de dados e este faz múltiplas leituras dos endereços da região 0xB0000000. No entanto, a melhor forma de o fazer é através da segunda opção, usando o *DMA (direct memory access)*. Este faz com que a transferência seja feita apenas de uma só vez para o processador, quando a transferência estiver completa, um *interrupt* de finalização é chamado.

A *Multichannel Systems* fornece um projecto de exemplo para o uso do *DSP* e deste tipo de transferências. Um registo importante para a leitura de dados é o 0x400, que controla que tipo de dados é transferido e pode permitir transferência de dados de diferentes *headstages* ou do *ADC* da *interface board*.

No manual deste sistema podemos encontrar todos os endereços mapeados pelo *DSP* no *MEA*, o que é bastante importante para fazermos configurações mais específicas quando necessário [49].

3.1.4.1 *DMA (Direct Memory access)*

Um *DMA* é um periférico capaz de transferir dados entre dois dispositivos mapeados em memória, mais concretamente permite que certos periféricos acedam à memória principal do sistema (RAM) independentemente do *CPU*.

Esta tarefa é de extrema importância no nosso sistema, pois permite-nos continuar a normal execução do programa do nosso processador enquanto faz a aquisição de dados em "paralelo"[50], assim que a transferência de um novo pacote de dados estiver terminada, então é gerado um *interrupt* de modo acedermos aos novos dados e podermos processá-los.

3.1.4.2 Organização dos Dados

Existem variadas formas de organizar os dados que vamos receber a partir do DSP, para isso temos que alterar a configuração dos registos de controlo do MEA2100[16][49].

No nosso caso, configurámos os registos de modo a que apenas recebamos os dados dos eléctrodos e o *timestamp* dos dados recebidos (funciona como um contador de sweeps).

Os dados são guardados no Vector MeaData, com no mínimo 260 posições, já que a primeira posição, *MeaData[0]*, está reservada para o *header*, *MeaData[1]* a *MeaData[256]* para a amostra actual de cada eléctrodo e as restantes três posições para o *timestamp*.

3.1.4.3 Conversão dos Dados

Os dados que recebemos no *DSP* foram quantizados pelo *ADC*, logo teremos que convertê-los novamente para o seu valor real em micro Volts.

O tipo de dados do *MeaData* é *unsigned int*, logo os valores de tensão positivos convertidos mantêm-se positivos, já os negativos transformam-se em positivos, sendo que teremos que subtrair o simétrico do seu valor ao valor máximo de um *unsigned int* que é de 2 elevado a 4*8, em que 4 equivale ao número de bytes deste tipo de dados.

Chegamos então à seguinte expressão, que nos diz o valor da largura de quantização (variação mínima da entrada detectável na saída do *ADC*):

$$F(Range, GAIN) = \frac{Range}{2^{ADCresol} \cdot GAIN} \tag{3.1}$$

Onde $F(x)$ denota:

GAIN =1000
Range = 2576 mV *2
ADCresol = 24 bits

3.2 Sistema de simulação do *MEA2100*

Nesta secção iremos descrever o *Kit* de desenvolvimento bem como o *DSP* contido nele, ao qual tivemos um acesso mais imediato para podermos desenvolver e validar os algoritmos de processamento sem necessidade de estar a ocupar o sistema *MEA2100*, partilhado por vários membros do grupo de investigação *NCN*.

3.2.1 LCDK-TMS320C6748

Em primeiro lugar é necessário explicar o porquê da necessidade do uso deste processador, já que na verdade a única coisa que podemos fazer com ele é simular dados e validar funções. O uso deste *Kit* mostrou-se de extrema importância em diversas fases do projecto, já que nem sempre temos acesso ao laboratório, devido à elevada demanda deste sistema por parte dos investigadores, e também quando tivemos problemas relacionados com peças de *hardware* que se estragaram, que nos impediram o acesso ao sistema. Assim, quando o *MEA2100* nos estava restringido pudemos avançar no desenvolvimento e validação de alguns algoritmos de detecção e também de comunicação com o computador.

O processador C6478 é a combinação entre os *64x+* e o *c67x+*[51] , e o código escrito para ele é totalmente compatível com o *DSP* apresentado acima, daí ter sido escolhido para desenvolver código sem o uso do *MEA2100*.

Este processador está embebido num *LCDK (low cost development kit)*, que nos oferece um grande número de periféricos para aquisição de dados, de modo a poderem ser realizados projectos de várias naturezas[52]. Dos quais serão utilizados apenas a porta série *USB-mini*, um *JTAG header*, portas *GPIO* para uso geral e LED's. Na figura 3.8 podemos os principais componentes deste *Kit*.

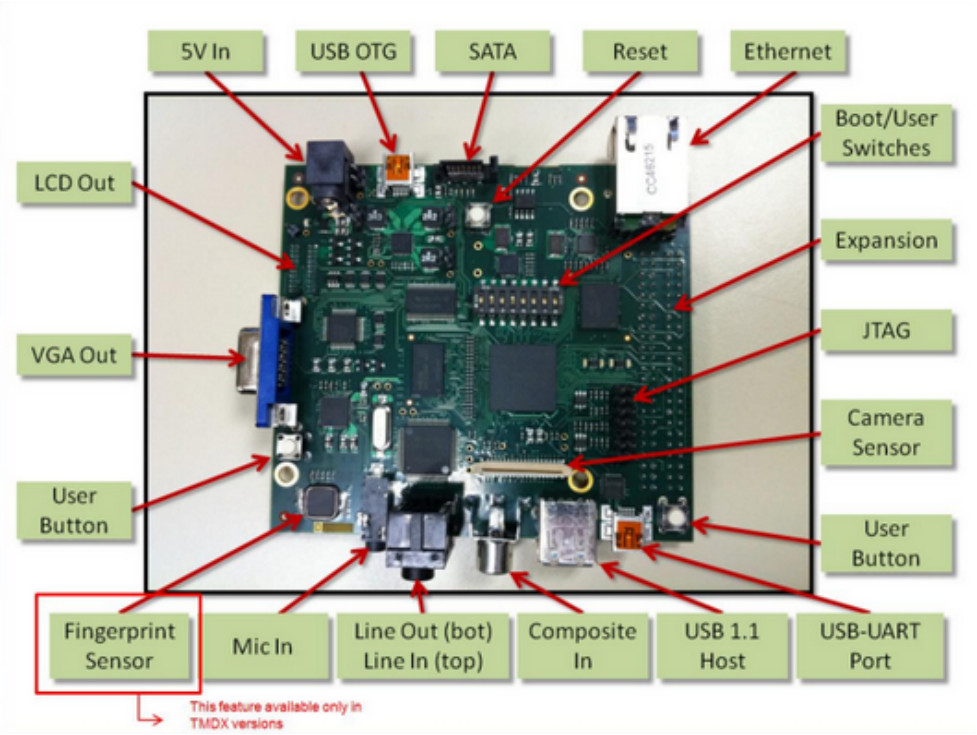

Figura 3.8: LCDK Hardware[52]

3.3 Conceito de sistema

A figura 3.9 representa de forma simplificada a estrutura e montagem do sistema que foi feito de modo a concretizar os objectivos do projecto.

No laboratório, a *headstage* encontra-se na base de um microscópio, de modo a que os investigadores possam visualizar as medições dos neurónios feitas pelo sistema e também observar os próprios neurónios em actividade. Os dados recolhidos na *headstage* são enviados para a *interface board* através do cabo *eSATA*. Através *interface board* fazemos as configurações de todo o *MEA2100* alterando os registos de configuração, quer seja através do *DSP* ou do computador, e fazemos também o processamento de sinal, enviando os dados obtidos para o computador sempre que este os requisitar.

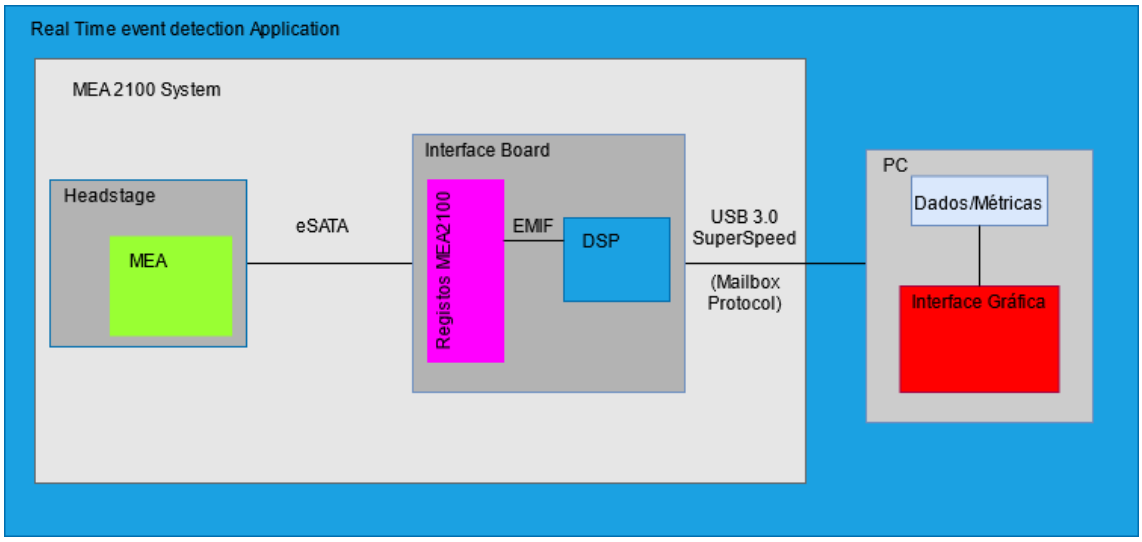

Figura 3.9: Conceito de Sistema

Já a figura 3.10 representa a montagem do sistema que pretende simular o que foi montado no laboratório para validação dos algoritmos. Assim, podemos ter uma ideia mais concreta da diferença entre ambos. Ao contrário do sistema *MEA2100* este não possui aquisição de dados, pelo que somos nós a criar os mesmos no *DSP*.

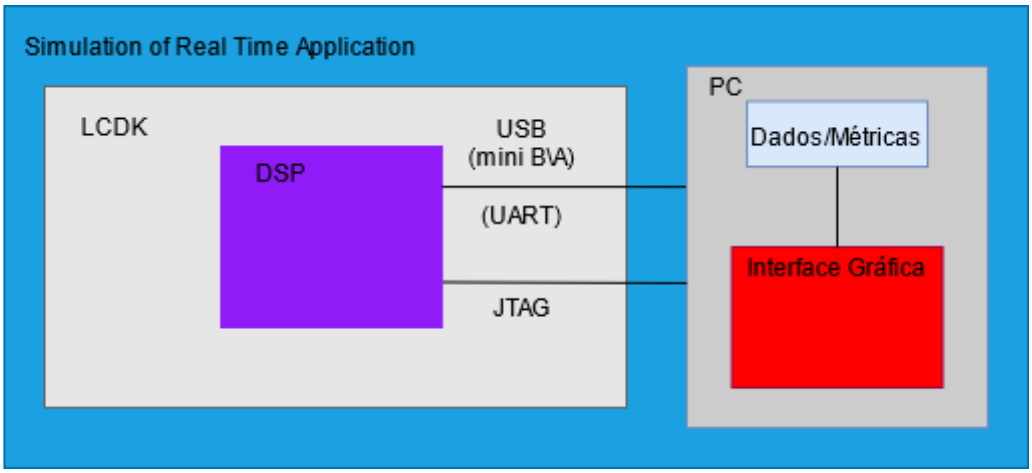

Figura 3.10: Sistema de simulação

3.4 Ferramentas utilizadas

Esta secção visa apresentar algumas das ferramentas necessárias para realizar a escrita do código e para fazer *debugging* , também algumas configurações necessárias à utilização do sistema *MEA2100*.

Decidimos colocar esta secção devido às dificuldades sentidas no início do projecto relativas a certas configurações e versões necessárias.

3.4.1 *JTAG*

O *JTAG (Joint Test Action Group)* é um circuito integrado capaz de realizar vários tipos de tarefa, como validar arquitecturas de sistemas integrados e o hardware neles contidos. No nosso caso este é usado apenas como ferramenta para fazer o *debug* do código escrito para o *DSP* através da *JTAG Interface* presente no *Code Composer Studio*. Este circuito permite-nos aceder às variáveis utilizadas a qualquer momento e fazer *breaks* no código, o que torna o *debugging* muito mais rápido e eficaz.

O *JTAG* que escolhemos utilizar é o *XDS100v2* da *Black Hawk*, que é um dos mais baratos e com mais qualidade para o efeito pretendido.

3.4.2 *Code Composer Studio (CCS)*

O *Code Composer Studio (CCS)* é o *IDE* que será utilizado, pois contém bastantes ferramentas para o uso de processadores da *TI* e para o *debug* dos mesmos.

Inicialmente, cometemos o erro de instalar a versão mais recente deste *IDE*, e tivemos inúmeros problemas na compilação de código nas plataformas c6748 e c6454, mais tarde, após falarmos com a equipa de suporte da *TI* fomos informados que para estes processadores teria que ser usada uma versão anterior, pelo que optamos pela versão 9 do *CCS*.

Deparámo-nos mais tarde com outro entrave, quando começámos a compilar código no c6454. Este processador deixou de ser comercializado, por conseguinte as versões dos compiladores mais recentes também não são compatíveis com este processador, por isso fizemos um *downgrade* do compilador para este processador. O c6748 utiliza a versão do compilador *TI v8.3.5*, já o c6454 a versão *TI v7.3.23*.

3.4.2.1 .cmd file

Este tipo de ficheiros são essenciais para a divisão de espaços de memória no *DSP*, que por vezes podem ter que ser alterados quando uma secção de memória deixa de ter espaço suficiente para o pretendido.

Os *Linker command files* podem conter instruções que são normalmente escritas na linha de comandos como opções, ficheiros objecto, bibliotecas, etc. O mais importante neste tipo de ficheiros é a directiva *MEMORY* e *SECTIONS*.

O objectivo da directiva *MEMORY* é atribuir nomes a espaços de memória, que depois são utilizado na directiva *SECTIONS*. Nesta são atribuídos a esses espaços de memória secções de código ou de variáveis. Existem formas de atribuirmos a uma secção um espaço de memória diferente dependendo do espaço requerido para essa secção.

Na figura 3.11 mostramos como está organizada a memória do *DSP* do *MEA2100-256*

```
MEMORY
{
        L1D:     o = 00f00000h   l = 00008000h  /* 32 KByte */
        L1P:     o = 00e00000h   l = 00008000h  /* 32 KByte */
        L2:      o = 00800000h   l = 000F0000h  /*  1 Mbyte */
        DDR:     o = 0e0000000h   l = 10000000h  /* 256 MByte */
}

SECTIONS
{
    .csl_vect   >       L2
    .text       >       L2
    .stack      >       L2
    .bss        >       L2
    .cinit      >       L2
    .cio        >       L2
    .const      >       L2
    .data       >       L2
    .switch     >       L2
    .sysmem     >       L2
    .far        >       L2
    .testMem    >       L2
    .intvecs    >       L2
     .dataddr    >       DDR
}
```

Figura 3.11: Organização da memória do *C6454*

3.4.2.2 *Interrupt Vector Table*

Uma *interrupt vector table* é uma estrutura de dados que associa uma lista de *interrupt handlers* com uma lista de *interrupt requests*. Cada *interrupt vector* corresponde a um endereço de um *interrupt handler*. Este define as prioridades dos *interrupts* e guarda-as numa fila de espera se mais do que um *interrupt* estiver à espera de ser manipulado. Existem várias formas de gerar um *interrupt*, normalmente através de periféricos ligados ao processador, que lhe dizem para interromper o que está a executar e decidir o que irá fazer a seguir. Quando um *interrupt* é gerado o processador guarda o estado de execução actual, e começa depois a executar o manipulador do *interrupt*.

O processador presente no *MEA2100-256* contém 128 tipos de eventos diferentes que podem ser associados para qualquer um dos doze *inputs* de *interrupts* do *CPU* (CPUINT4 - CPUINT15). No nosso caso iremos usar apenas três deles.

3.4.3 Linguagem de Programação

A linguagem de programação utilizada para programar o DSP é *C*. Já no caso do desenvolvimento da interface gráfica é utilizado *C#*, com a ajuda do *Windows Forms*, que é uma *framework* com interface gráfica que facilita a elaboração de *Desktop Apps*. Sendo o *IDE* utilizado o *Visual Studio Code*. Foi também utilizado o *MATLAB* para a realização de *scripts* para fazer validação de algoritmos.

3.4.4 Bibliotecas

De modo a sermos capazes de desenvolver código nos processadores/controladores, a *TI* oferece-nos ferramentas que facilitam a inicialização dos periféricos e utilização do seu sistema operativo em tempo real, e também bibliotecas de algoritmos optimizados. Temos então como ferramentas o *Starter Ware*, conjunto de ferramentas para desenvolvimento de programas em *baremetal*, ou seja, sem sistema operativo, ou então o *Processor SDK*, que nos oferece um grau maior de abstracção bem como um sistema operativo (TI-RTOS) integrado, no caso de ser necessária a execução de várias tarefas ao mesmo tempo pelo processador.

Decidimos fazer alguns programas exemplos com ambas as ferramentas para conseguirmos perceber qual destas seria a mais indicada para fazer a simulação do nosso sistema. Começamos por utilizar o *Processor SDK*, no entanto este demonstrou ter um nível de abstracção que não nos ajudava na nossa aplicação, já que não podíamos manipular com muita transparência os dispositivos de hardware presentes no kit. Por isso, passamos para o *Starter Kit*, que foi bastante mais fácil de utilizar e identificar os ficheiros necessários para o processo de *Linking*.

Identificámos mais tarde outro problema. Devido ao facto do c6454 ser um processador mais antigo as bibliotecas que são utilizadas neste processador são diferentes nas utilizadas no *LCDK*, por isso ao transpor o código de um processador para o outro temos que ter em atenção esta situação.

3.4.4.1 *MCS DLL(Dinamic Link Library)*

Uma *DLL* é uma biblioteca, contendo código e dados, que pode ser usada por mais do que um programa ao mesmo tempo. Este tipo de bibliotecas ajuda a promover a eficiência do uso de memória e a reutilização de código[2].

Tendo isto em mente, a *MCS* fornece uma *DLL* que providencia uma interface *.NET* com os dispositivos da *MCS*. As ferramentas mais importantes que contém são as de acesso aos dispositivos de aquisição de sinais e ao estimulador. É possível, por isso, que se façam programas em diversas linguagens de programação para análise de dados neuronais.

No nosso caso em concreto será essencial para o envio dos dados recolhidos pelo *DSP* para o computador e também para que o utilizador possa configurar parâmetros de detecção e que canais quer monitorizar através de uma interface intuitiva.

[2]https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library

Capítulo 4

Processamento DSP

Neste capítulo apresentamos, na primeira secção, que estruturas de dados foram utilizadas para guardar toda a informação necessária. A segunda secção apresenta os algoritmos desenvolvidos para fazer a análise, em tempo real, dos sinais neuronais.

Na terceira secção, analisamos como foi estruturado o código no *LCDK*, que contém o processador C6748 e que comunicará com o computador enviando métricas que o utilizador pretenda. Este sistema, *LCDK/PC*, pretende emular o sistema *MEA2100/DSP-C6454/PC* de modo a podermos validar as funções criadas para análise de sinais neuronais. Assim, em vez de termos o *MEA2100* a fazer aquisição de sinal, temos o *LCDK* a simular sinais de neurónios, seguida da sua análise em tempo real no *DSP* incluído no kit, e por fim o envio dos eventos para o *PC* através da *User Interface* desenvolvida. Com esta plataforma podemos desenvolver os algoritmos necessários sem estarmos dependentes do uso do sistema real (*MEA2100-256*) que tem uma disponibilidade bastante reduzida.

Por fim, na última secção, iremos fazer algo análogo à secção anterior, mas desta vez referente ao código presente no *MEA2100-256*.

4.1 Variáveis

As variáveis mais importantes neste contexto serão as referentes aos dados que pretendemos obter da leitura dos sinais neuronais, o nosso temporizador.

Temos um vector com 256 posições, *MeaData*, que guarda os dados, vindos da *headstage*, sendo que quatro destas posições são massas e não devem ser monitorizados. Este vector é actualizado a cada nova *frame* de dados recebida. Os valores recebidos pelo *DSP*, adquiridos da *headstage*, presente no *MEA2100* são do tipo *unsigned int*, por isso decidimos que o melhor seria replicar este tipo de dados também na simulação. Estes dados recebidos são valores quantizados pelo *ADC*, por isso não representam directamente o valor, em volts, que o potencial dos neurónios apresentam, para tal teremos que fazer a conversão destes dados. Existe, também, outro vector

que nos diz que eléctrodos serão monitorizados, sendo estes determinados pelo utilizador. Torna-se importante este vector já que a ordem em que recebemos e gravamos os dados no *MeaData* não é sequencial devido à arquitectura do próprio *MEA*.

Temos uma estrutura de dados para fazermos a contagem de tempo que passa à medida que cada nova *frame* de dados é recebida, ou criada, no caso da simulação com o *LCDK*. Como veremos na secção 4.4, para o MEA o temporizador é criado de forma diferente. Criámos também uma estrutura para fazermos *Spike Detection*, outra para *Burst Detection* e para *Networkburst Detection*. A estrutura *Spike* contém o número do canal a que pertence, um contador de *spikes*, um vector com os instantes a que cada *spike* ocorreu. A estrutura *Burst* também contém o número do canal a que pertence, um contador de *bursts*, uma variável que nos diz quantos *spikes* contém um *burst*, um vector que guarda em que instantes foi detectado, uma variável que indica qual é o tempo máximo entre *spikes* consecutivos para que estejamos ainda em *burst* e uma *flag* que nos informa se o eléctrodo em causa se encontra ou não em *burst*. Já a estrutura *Network Burst* só terá um elemento, pelo que não necessita de uma variável que guarde o canal a que pertence. Assim, tem apenas uma variável que guarda o tamanho do *network burst*, que dita quantos canais se encontram em *burst* a cada instante, outra que guarda quantos *network bursts* ocorreram e um vector que guarda em que instantes é que estes ocorreram. A figura 4.1 mostra as estruturas descritas em cima.

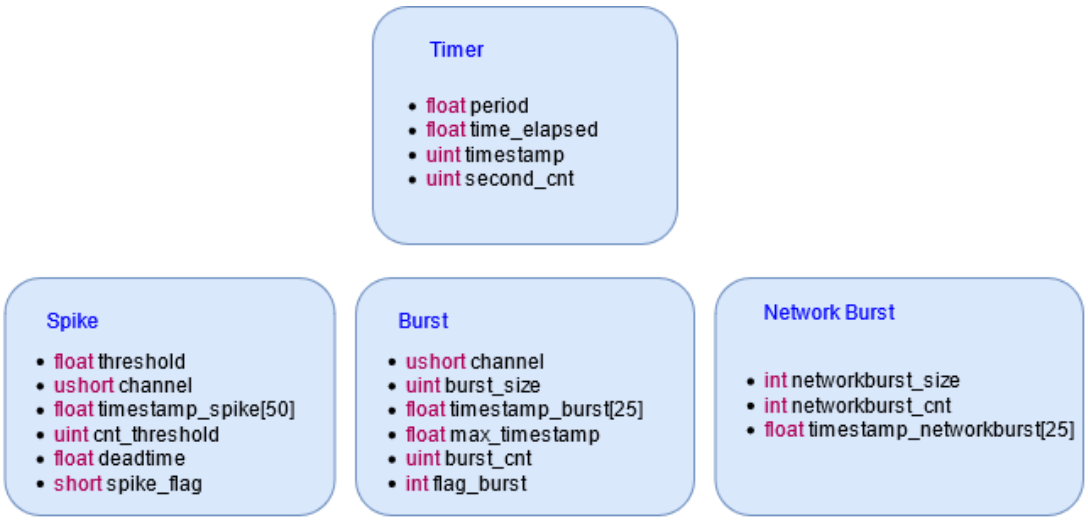

Figura 4.1: Estruturas de dados

É de notar que os vectores que guardam os tempos em que ocorreram os eventos têm um tamanho limitado, já que a memória do *DSP* é reduzida (256 KB), portanto temos que ter em consideração que quando estes são preenchidos totalmente, temos que começar a escrever por cima dos dados anteriores a partir da primeira posição. É claro que ainda temos alguma margem de memória que poderíamos utilizar para que estes vectores fossem maiores, no entanto temos

que ter em consideração que todo o código escrito neste trabalho será a base de muitas outras experiências que poderão serão feitas por parte do laboratório, por isso será importante que esta margem seja suficientemente grande para tal.

4.2 Algoritmos

Nesta secção analisamos como foram implementados os algoritmos de detecção de eventos neuronais, nomeadamente: *spikes*, *bursts* e *network bursts*. É desde já importante referir que iremos mostrar cada algoritmo apenas como se um canal fosse monitorizado. Na verdade foi assim que elaborámos estes algoritmos, que estarão dentro de um ciclo *for* que percorre todos os canais que o utilizador decidir monitorizar no *MEA*, conforme representado esquematicamente na figura 4.2.

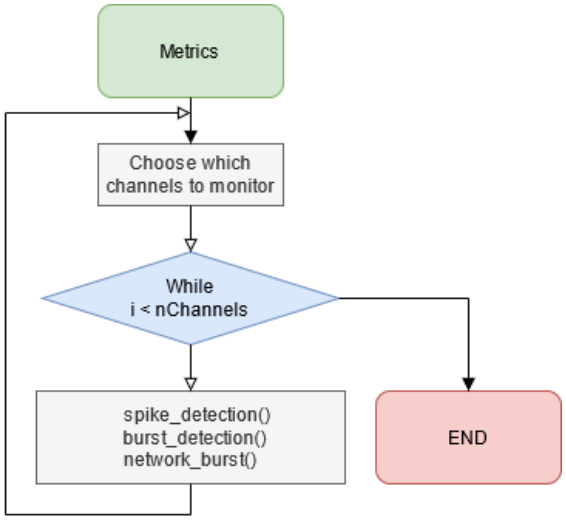

Figura 4.2: *Loop* de detecções

Os algoritmos que vamos mostrar são elaborados com base no uso das estruturas de dados que foram criadas para a detecção dos eventos, estruturas estas que estão presentes na secção anterior.

4.2.1 Detecção de *Spikes*

Esta função é a chave para todas as outras métricas, todas vão precisar da informação proveniente desta, por isso merece especial atenção. Um *spike* é detectado sempre que o valor de tensão de um dado canal for superior e o anterior inferior a um certo *threshold*.

O algoritmo de detecção de *spikes* é chamado a cada iteração do *interrupt* após um novo conjunto de dados ter sido recebido.

Na função de *spike detection* é feita a conversão dos dados quantizados pelo *ADC* para micro *Volts*, que passam de dados do tipo *unsigned int* para *float*, sendo que, depois de convertidos, estes valores tanto podem tomar valores tanto negativos como positivos. Depois, é feita uma condição,

no caso de um desses eléctrodos ter um potencial que ultrapasse o *threshold*, positivo ou negativo, que pode ser escolhido pelo utilizador, e o seu valor anterior tiver sido menor (no caso de potencial positivo) que o actual é detectado um *spike*. No entanto, existe uma segunda condição para que o *spike* seja válido, apenas se a diferença de tempo entre o *spike* actual e o anterior for maior que um certo valor de tempo (*deadtime*) é o *spike* válido, para prevenir que o mesmo *spike* seja detectado mais do que uma vez, devido à existência de ruído de alta frequência ou um spike bifásico, isto é, com um pico positivo e negativo. .

Após a detecção de um *spike* num dado eléctrodo, é guardado num vector, com tamanho limitado, em que instante este ocorreu, sendo esta informação depois enviada para o computador.

Na figura 4.3 está representado um diagrama de fluxo simplificado da forma como este algoritmo se comporta:

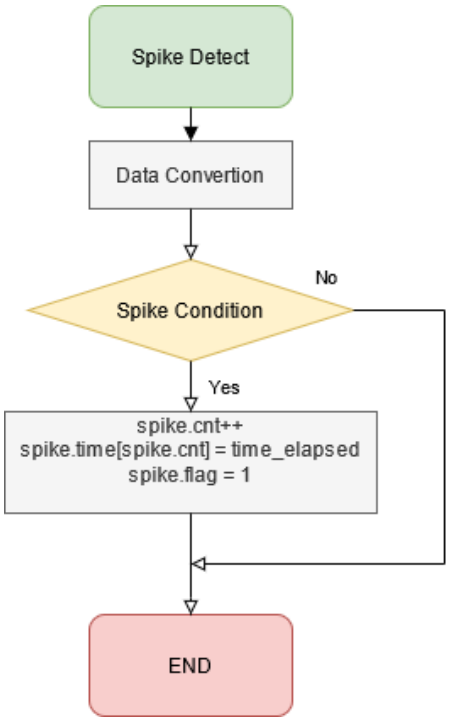

Figura 4.3: Algoritmo de Detecção de *Spikes*

- **Data Convertion:** Neste bloco é feita a conversão dos dados para valores de tensão dos neurónios

- **Spike Condition:** No caso do potencial de acção ser positivo, a amostra anterior deve estar abaixo do *threshold* e a actual acima. No caso de potencial de acção negativo, a amostra anterior deve ser maior que o *threshold* (negativo) e a actual menor, para que seja detectado o *spike*. Uma segunda condição obriga a que todos estes casos obedeçam à condição do *deadtime*, só é detectado um *spike* se já tiver passado no mínimo esse tempo desde a detecção anterior.

4.2.2 Detecção de *Bursts*

O algoritmo de detecção de *bursts* é chamado a cada iteração do *interrupt*, após o algoritmo de detecção de *spikes* dentro do mesmo ciclo. Foram feitos dois algoritmos diferentes, sendo que um deles executa a detecção mais rapidamente que o outro, por isso foi o escolhido para a tarefa.

Um *burst* é contabilizado quando pelo menos mais do que um número mínimo de *spikes*, num dado canal, é detectado, sendo que o intervalo de tempo entre estes *spikes* tem que ser inferior a um certo valor definido pelo utilizador.

No primeiro algoritmo criado gerámos uma janela temporal a partir do momento em que era detectado um *spike* e se dentro dessa janela temporal fossem detectados mais que o número de *spikes* mínimo, então teríamos um *burst*, que terminaria quando o intervalo de tempo entre o *spike* anterior e o actual fosse maior do que um certo valor definido pelo utilizador.

Depois de analisarmos o algoritmo anterior chegámos à conclusão que existia uma forma mais simples de o executar. O que ele faz é verificar se existe algum canal em que foi detectado um *spike*, se sim é iniciado um contador de tamanho de *burst*, se o *spike* seguinte tiver ocorrido dentro de um certo intervalo de tempo menor do que o tempo máximo admissível, escolhido pelo utilizador, então o tamanho do *burst* é incrementado. Quando o tempo que passou desde o último *spike* for maior do que o tempo máximo admissível, então é verificado o tamanho do *burst*, se este tamanho for maior do que o tamanho mínimo de *burst*, escolhido pelo utilizador, então esse *burst* é contabilizado.

Na figura 4.4 representamos num diagrama de fluxo o algoritmo simplificado feito por nós:

Figura 4.4: Algoritmo de Detecção de *Bursts*

4.2.3 Detecção de *Network Bursts*

Tal como os algoritmos anteriores, este é também chamado dentro do ciclo que percorre todos os canais que estão a ser monitorizados.

Um *network burst* é contabilizado quando existe pelo menos mais do que um número mínimo de canais em *burst* simultaneamente, este número mínimo é definido pelo utilizador. Este algoritmo está dividido em duas partes distintas.

A primeira parte verifica se o canal em causa se encontra em *burst*. Se este *burst* ainda não tiver sido contabilizado para o *network burst* a decorrer, o tamanho do *network burst* é incrementado e o canal é indicado como contabilizado através de uma *flag*. Caso o canal já tenha sido contabilizado e já não se encontrar em *burst*, então o tamanho do *network burst* é decrementado.

Na segunda fase verificamos se o número de eléctrodos em *burst* é maior ou igual ao número mínimo necessário para ser considerado um *network burst*. Assim que o *network burst* seja detectado, o seu tempo é guardado, o contador de *network bursts* é incrementado e é assinalado que existe um *network burst* a decorrer através de uma *flag*. Caso o tamanho do *network burst* seja menor que o o seu valor mínimo, a *flag* que nos indica se estamos ou não perante um evento deste tipo é colocada a zero.

Na figura 4.5 podemos verificar como foi implementado o algoritmo:

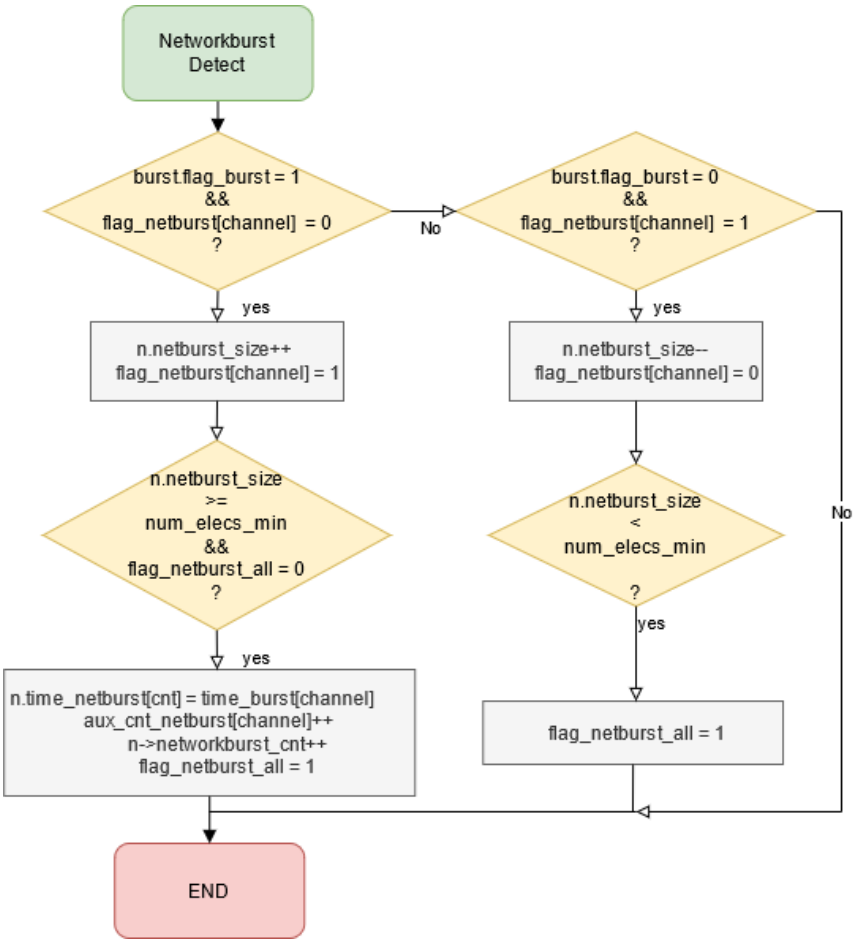

Figura 4.5: Algoritmo de Detecção de *Network Bursts*

4.3 Kit

4.3.1 Estrutura do código DSP

Esta secção foca-se no modo como foi pensado e estruturado o código para simular o *MEA2100*. Podemos então dividi-lo em duas partes principais: a geração de dados que simulam a actividade neuronal e a execução das funções de detecção descritas na parte anterior.

4.3.1.1 Função *Main*

Antes de entrarmos no *main loop*, temos que fazer a configuração dos periféricos utilizados, os *LED's*, o *UART*, o *power sleep controller*, os *timers* e também as configurações dos *interrupts* que serão utilizados.

Após ter sido feita a configuração falta apenas definir que eléctrodos deverão ser monitorizados e quais são os eventos a serem visualizados. Isto é feito pelo utilizador através da interface

gráfica, este escolhe quais serão os eléctrodos que quer visualizar e que medições quer fazer, posteriormente esta informação é enviada para o processador.

Após todas estas configurações estarem efectuadas entramos num ciclo infinito, no qual os *interrupts* controlam quando serão efectuadas as suas tarefas, sabendo que os *interrupts* de numeração inferior têm uma prioridade mais elevada, por isso podem interromper a execução de outro com prioridade inferior. Na figura 4.6 podemos ver a estrutura da função *main*.

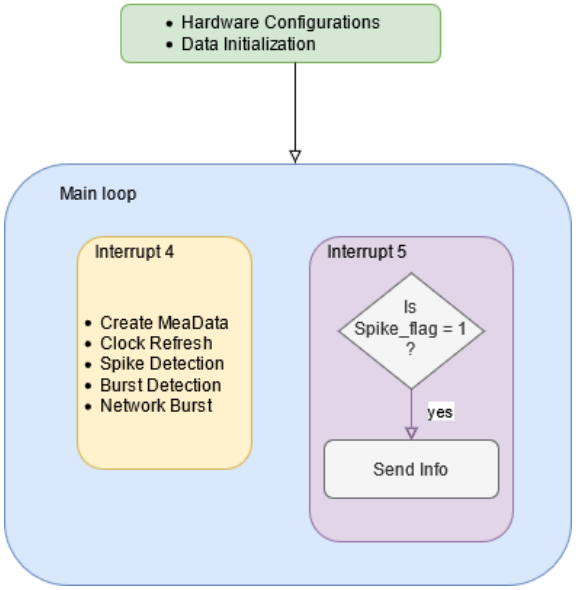

Figura 4.6: Estrutura da função *Main*

4.3.1.2 *Main loop*

Depois de as configurações terem sido feitas o programa entra num *loop* infinito. Para recriarmos o sistema real com o máximo de aproximação possível ao real, criámos duas *callback functions* que são chamadas pelos respectivos *interrupts*. Estes *interrupts* são controlados através de temporizadores independentes, quando cada um destes chega ao tempo definido é então chamada a respectiva função.

A função com a prioridade mais alta, chamada pelo *interrupt 4*, contém o código que gera o *MeaData*, tal como no sistema real, em que é gerado um *interrupt* quando são adquiridos novos dados. Ainda dentro desta função, o nosso relógio é actualizado e, consoante o tipo de métricas que o utilizador quer observar, são chamadas as funções de detecção de *spikes* e *bursts*.

A outra função serve apenas para a comunicação com o computador, mais uma vez sendo chamada periodicamente, pelo *interrupt 5*, se tiver sido detectado um *spike/burst* esta informação é enviada para o computador, no caso de existirem muitos *spikes* de uma só vez podemos ter uma latência indesejada no programa. No sistema real não será feito o envio desta exacta forma, pois o protocolo de comunicação é diferente e terá que ser o computador a pedir periodicamente estas

métricas (usando o método de *Polling*), funcionando como um sistema em que o *PC* é o *master* e o *DSP* o *slave*.

4.3.2 Validação dos Algoritmos

Numa fase inicial, limitámo-nos a gerar um número aleatório para cada eléctrodo, através da função *rand()*, dentro de um certo limite (p.e. 0-15000), a cada iteração do nosso *interrupt 4*, de modo a percebermos se os dados estavam a ser transferidos de forma correcta do DSP para o PC, comparando o valor dos dados gerados com os valores recebidos pelo computador.

Para fazermos a validação dos algoritmos apresentados na secção 4.2 criámos uma função que simula os dados recolhidos do *MEA2100*, ou seja simulámos os dados das células neuronais.

Esta função apresenta vários modos, em que cada um deles apresenta comportamentos diferentes ao longo do tempo, de modo a simularmos diferentes situações que poderão ocorrer com os dados recebidos das células neuronais. Para a simulação da actividade neuronal fizemos uma simplificação na sua forma de onda, pois a forma de onda do potencial dos neurónios é bastante irregular e imprevisível, bem como contém bastante ruído, o que consumiria muito tempo a ser simulado. Com isto, como sabemos que apenas os valores que ultrapassam o *threshold*, positivo ou negativo, serão contabilizados como *spikes*, limitámo-nos a criar esses *spikes*, ou seja, nos instantes de tempo definidos nessa função serão criados pulsos, que representam os *spikes*.

Por fim, para validarmos os resultados, os tempos em que ocorreram *spikes* serão enviados via *USB-UART* para o computador, e estes dados serão gravados pelo nosso código *C#* e, posteriormente, serão analisados por um *script* em *MATLAB* que irá mostrar-nos os *spikes* que foram detectados (de notar que o valor do potencial dos *spikes* está normalizado para uma visualização mais simples) e um *interspike histogram*, ferramenta muito utilizada nesta área, para sabermos a distribuição de intervalos de tempo entre *spikes* consecutivos.

4.3.2.1 Modo A: quatro *spikes* seguidos de intervalo

Neste modo são criados, para todos os canais, quatro *spikes*, positivos, a cada 0,02 segundos, seguidos de uma pausa de 0,04 segundos até ao próximo *spike*, depois repete-se o ciclo. Na figura 4.7 podemos observar os gráficos obtidos no *matlab* para um dos canais.

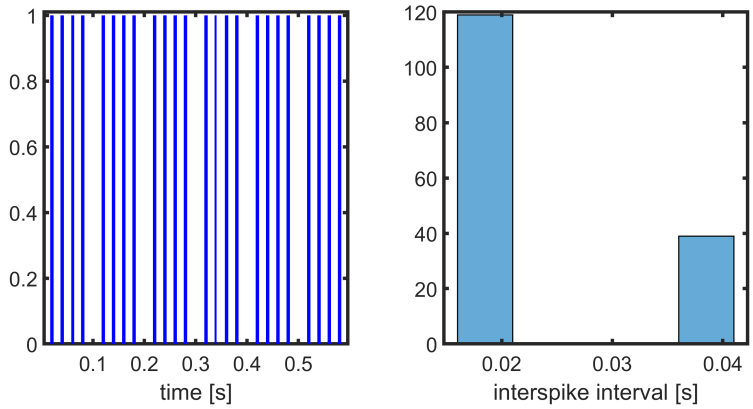

Figura 4.7: Modo A

Com esta validação podemos perceber que os *spikes* estão a ser gerados e detectados correctamente. No *interspike histogram* podemos ver que a proporção do número de spikes detectados nos intervalos de tempos já referidos está correcto, sendo este de 3 para 1.

4.3.2.2 Modo B: *spikes* de segundo a segundo positivos e negativos

Neste modo criámos os dados de um em um segundo, sendo que alternadamente os valores de cada *spike* são negativos e positivos, podemos assim confirmar que a função funciona tanto para *thresholds* positivos e negativos. Para além disso estamos a testar se o valor de *deadtime* está a ser respeitado, isto é, se não são detectados *spikes* consecutivos com um intervalo de tempo menor que o valor definido como *deadtime*, neste caso definido a 0.003s. Para isso, ao criarmos os *spikes* de segundo a segundo são gerados outros dois 0,0020 e 0,0024 segundos depois do primeiro, e como podemos observar na figura 4.8 apenas é detectado o primeiro desses *spikes*, sendo que os outros não são contabilizados.

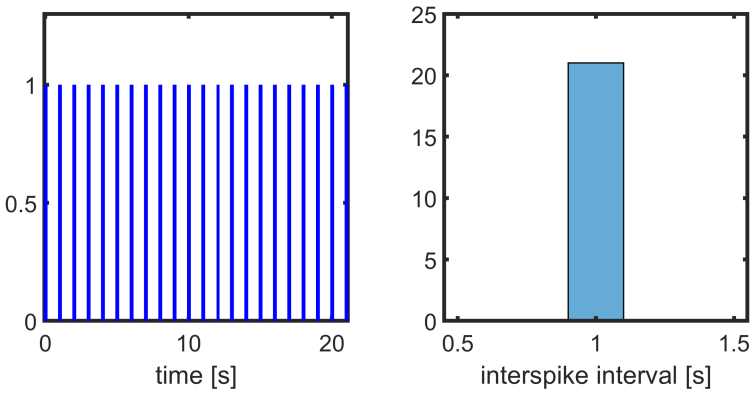

Figura 4.8: Modo B

4.3.2.3 Modo C: intervalos gaussianos

Depois de validarmos as funções com os valores gerados da forma supra mencionada, pensámos que talvez precisássemos de um ambiente menos controlado para detectarmos possíveis bugs no código, já que os valores eram gerados em intervalos determinados por nós, podiam estar a escapar-nos possibilidades que não tínhamos previsto.

Então, decidimos que em vez de ditarmos em que instante ocorria um *spike*, que este instante fosse determinado de cada vez que um novo *spike* ocorresse, ou seja, quando um *spike* é detectado é determinado quanto tempo falta para ocorrer o próximo através de uma distribuição normal. Assim, ao fazermos o nosso *interspike histogram* teremos que obter um histograma com a forma de uma curva gaussiana. Esta curva deverá ter a média e desvio-padrão correspondentes ao que foi predefinido na função que gera os *spikes*.

Na teoria das probabilidades a distribuição normal ou Gaussiana é um tipo de distribuição de probabilidades contínua para um valor aleatório no domínio real.

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{4.1}$$

O parâmetro μ corresponde à média da distribuição e o parâmetro σ ao desvio padrão.

Após a ocorrência de um *spike*, utilizamos o método *Box-Muller* para determinar qual deveria ser o intervalo de tempo até a geração do próximo *spike*. O método de *Box-Muller*, tal como todos os outros algoritmos, para gerar estas distribuições, necessita de um gerador de números aleatórios capaz de produzir valores aleatórios uniformes. Este método utiliza dois valores independentes distribuídos uniformemente entre 0 e 1.

$$X = \sqrt{-2ln(U)}cos(2\pi V) \tag{4.2}$$

Começámos por definir a média da nossa distribuição normal, que vai corresponder ao intervalo de tempo em que o próximo *spike* ocorrerá, como 1000 (*timestamp*), que corresponde a um tempo de 1000*0.00002s = 0.02s, e o desvio padrão como 300, que corresponde a 0.006s.

Na figura 4.9 podemos observar que o intervalo de tempo entre *spikes* que ocorreu mais vezes foi exactamente o que tínhamos previsto, 0.02s. Sendo que no *MATLAB* podemos confirmar que a média destas diferenças é de facto 0.02s e o desvio padrão ronda os 0.006s, não sendo exactamente igual devido à pequena quantidade de dados medidos.

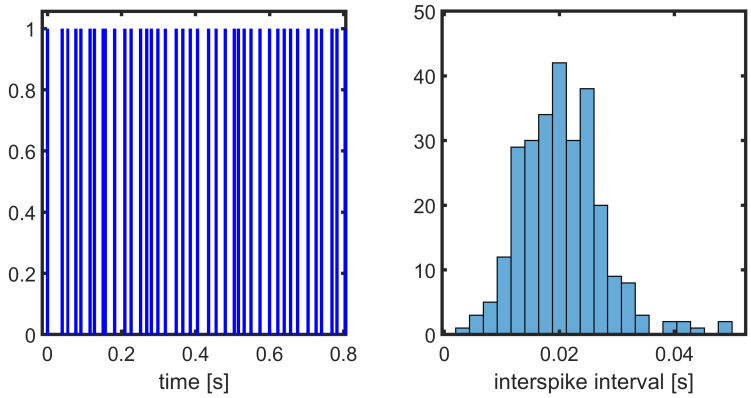

Figura 4.9: Modo C - período médio de 0.02s

Posteriormente, fizemos o mesmo estudo com uma média e desvio padrão diferentes, de 1500 (0.03s) e 400 (0.008s) respectivamente. Na figura 4.10 podemos verificar que a média mais uma vez bate certo com o previsto e também que o aumento do valor de desvio padrão se traduziu numa maior dispersão dos intervalos gerados em relação ao caso anterior. No *MATLAB* conseguimos verificar que o valor da média das diferenças bate certo com o espectável, já o desvio padrão está ligeiramente diferente ao que era esperado, mas ainda assim não significativo.

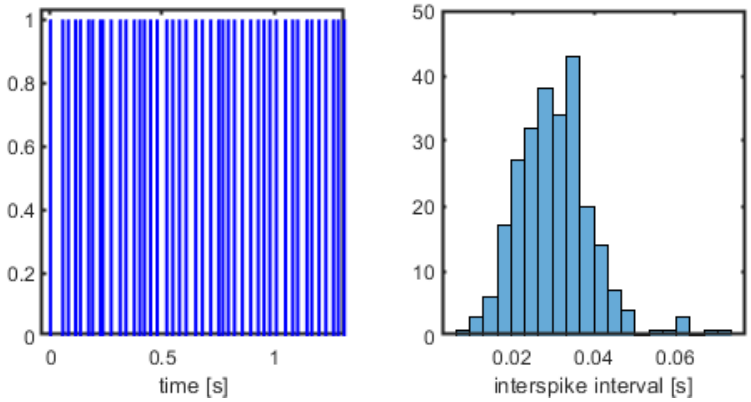

Figura 4.10: Modo C - período médio de 0.03s

4.4 MEA

4.4.1 Estrutura do código DSP

Neste secção mostramos como está organizado o código presente no processador, *c6454*, que se encontra na *interface board* do *MEA2100*, de forma análoga ao que foi feito na secção anterior.

4.4.1.1 Função *Main*

Para realizarmos as configurações do *MEA2100* foi necessário recorrer ao *MEA2100-256 User Guide*, no qual podemos ver todos os registos de configuração disponíveis.

Como podemos observar na figura 4.11 existe uma alteração na estrutura do código face à que foi feita no *LCDK*. Começámos por organizá-lo da mesma forma, ou seja, com apenas dois *interrupts*. Nos quais, um deles, com a prioridade mais elevada, era chamado quando uma nova *frame* de dados estava disponível, aí actualizávamos o relógio e fazíamos a detecção de eventos, já o segundo *interrupt* ocupa-se do envio dos dados. No entanto, no decorrer das validações foi necessário um reajuste desta estrutura, na qual foi introduzido mais um *interrupt* que serve para fazermos a actualização do relógio separadamente.

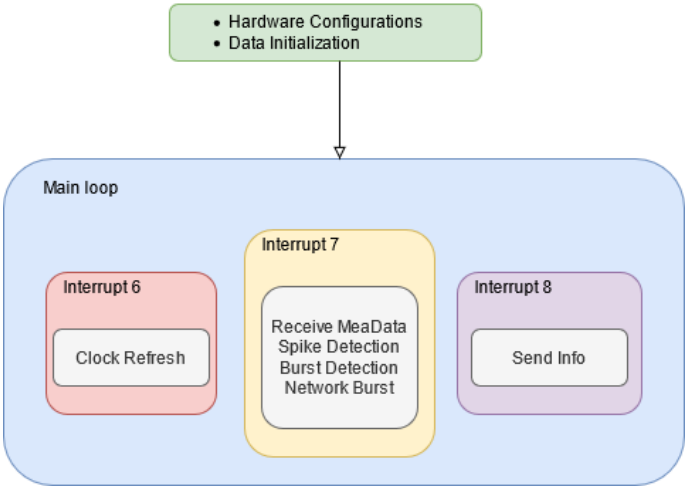

Figura 4.11: Estrutura da função *Main*

4.4.1.2 *Main Loop*

A execução do programa quando começámos a realizar as primeiras validações com apenas dois canais parecia estar operacional e com bons resultados, no entanto, quando aumentámos o número de canais gradualmente, até a um limite de cerca de vinte eléctrodos, começaram a surgir problemas. Continuamos a ser capazes de ler todos os eventos à medida que eles eram gerados, no entanto os instantes em que eles ocorriam, segundo os dados do *DSP*, não batiam certo com os que estavam a ser criados por parte do *Experimenter*, o que gerou alguma confusão. Chegámos à conclusão, depois de alguns testes, de que o problema se devia ao facto de, por vezes, o tempo de execução da detecção de eventos, quando existem vários canais a detectarem eventos em simultâneo, ser maior do que o período de tempo a que aquele *interrupt* é chamado. O nosso relógio não estava a actualizar tão rapidamente quanto devia, no período que tínhamos previsto de 0.00002s (50 KHz), fazendo com que os instantes em que ocorriam os eventos parecessem ocorrer a uma cadência mais rápida pelo *DSP*. Tivemos, então, que arranjar uma forma de resolver este problema.

Poderíamos tentar tornar as detecções mais eficientes, no entanto também não nos garantia que com um certo número de eléctrodos deixasse de funcionar. A ideia que nos pareceu mais conveniente para resolver esta situação foi a de diminuirmos a taxa de amostragem dos dados, já que o *interrupt* iria ser chamado menos vezes teríamos tempo suficiente para realizarmos todas as tarefas necessárias. Tentamos, por isso, mudar a configuração da taxa de amostragem a partir do registos de configuração do *MEA2100*, no entanto verificamos que a taxa de aquisição do sistema é fixa (50kHz). Finalmente, partimos para a solução final, que consiste em arranjar uma forma de tornar o nosso relógio independente da amostragem do sinal. Para tal, críamos mais um *interrupt* que deverá ser chamado periodicamente, ou seja, um *timer*, no qual o seu período equivale à cadência a que actualizamos o relógio. No entanto, temos que ter cuidado para que este não tenha uma cadência demasiado rápida, pois assim também entrará em conflito com a aquisição de dados. Tentamos posteriormente colocar tanto a actualização do relógio como a detecção de eventos no mesmo *interrupt* independente do *interrupt* de aquisição de dados, de modo a que fizéssemos a aquisição a 50kHz e paralelamente a uma frequência de 10kHz actualizar o relógio e correr os algoritmos, no entanto não obtivemos a performance pretendida por uma razão que não conseguimos apurar.

Assim sendo, a solução final consiste na utilização não de dois, mas de três *interrupts*, no qual o que tem prioridade mais elevada é o *interrupt* que faz a actualização do nosso relógio, a segunda prioridade fica reservada para a aquisição de dados e detecção de eventos e o terceiro para o envio dos mesmos.

4.4.2 Validação dos Algoritmos

As funções *Spike Detection*, *Burst Detection* e *Network Detection* desenvolvidas no *LCDK* foram depois testadas no *DSP* do *MEA2100*. Para tal, foi necessário o uso do *software* da *Multi-Channel Systems*, *MultiChannel Experimenter*, que nos permite criar *spikes* artificiais nas leituras dos *MEAs* aplicando estímulos eléctricos em eléctrodos específicos. Estes estímulos podem ser gerados de forma modular, criando ciclos com o número de *spikes* que queremos, à cadência que queremos. Na figura 4.12 podemos ver como são gerados. Para além disto, fizemos também alguns *scripts* de *MATLAB* para validar os dados obtidos, tanto directamente vindos *DSP*, retirados do *CCS*, como os que são enviados para o computador e gravados em ficheiros.

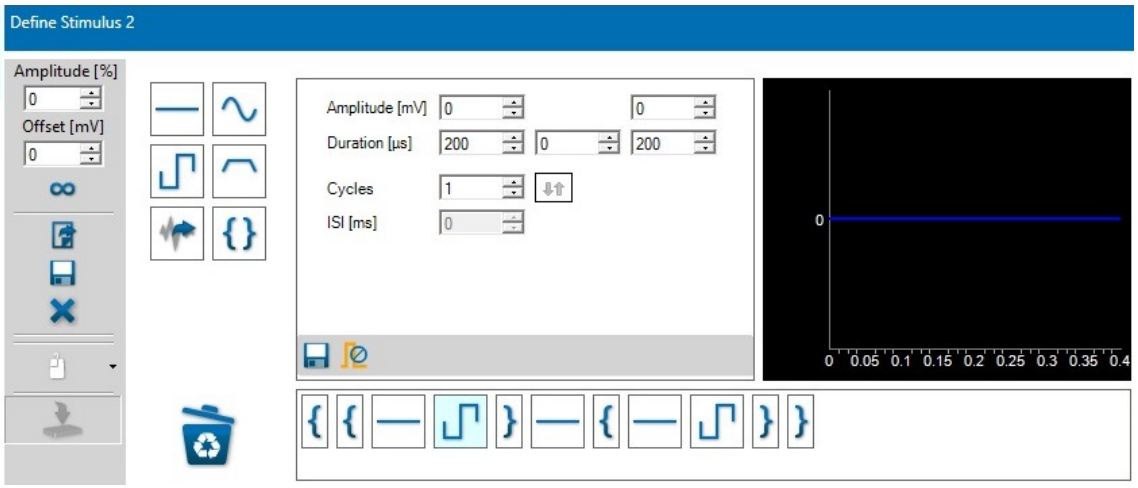

Figura 4.12: Criação de estímulos no *MultiChannel Experimenter*

4.4.2.1 Validação de *Spike detection*

Nesta primeira validação limitámo-nos a criar estímulos num eléctrodo que está próximo de outros dois eléctrodos que estarão a ser monitorizados, de modo a que esse estímulo ultrapassasse o valor de *threshold* definido, com o objectivo de observarmos se os estímulos gerados são os mesmos que são detectados pela função de *spike detection*.

O tempo em que ocorreram os *spikes* foram exportados do *Code Composer Studio* e foi também exportada a gravação feita no *MultiChannel Experimenter*. Através de um *script* realizado em *Matlab* verificámos que os *spikes* detectados correspondem aos estímulos gerados por nós, conforme mostrado na figura 4.13.

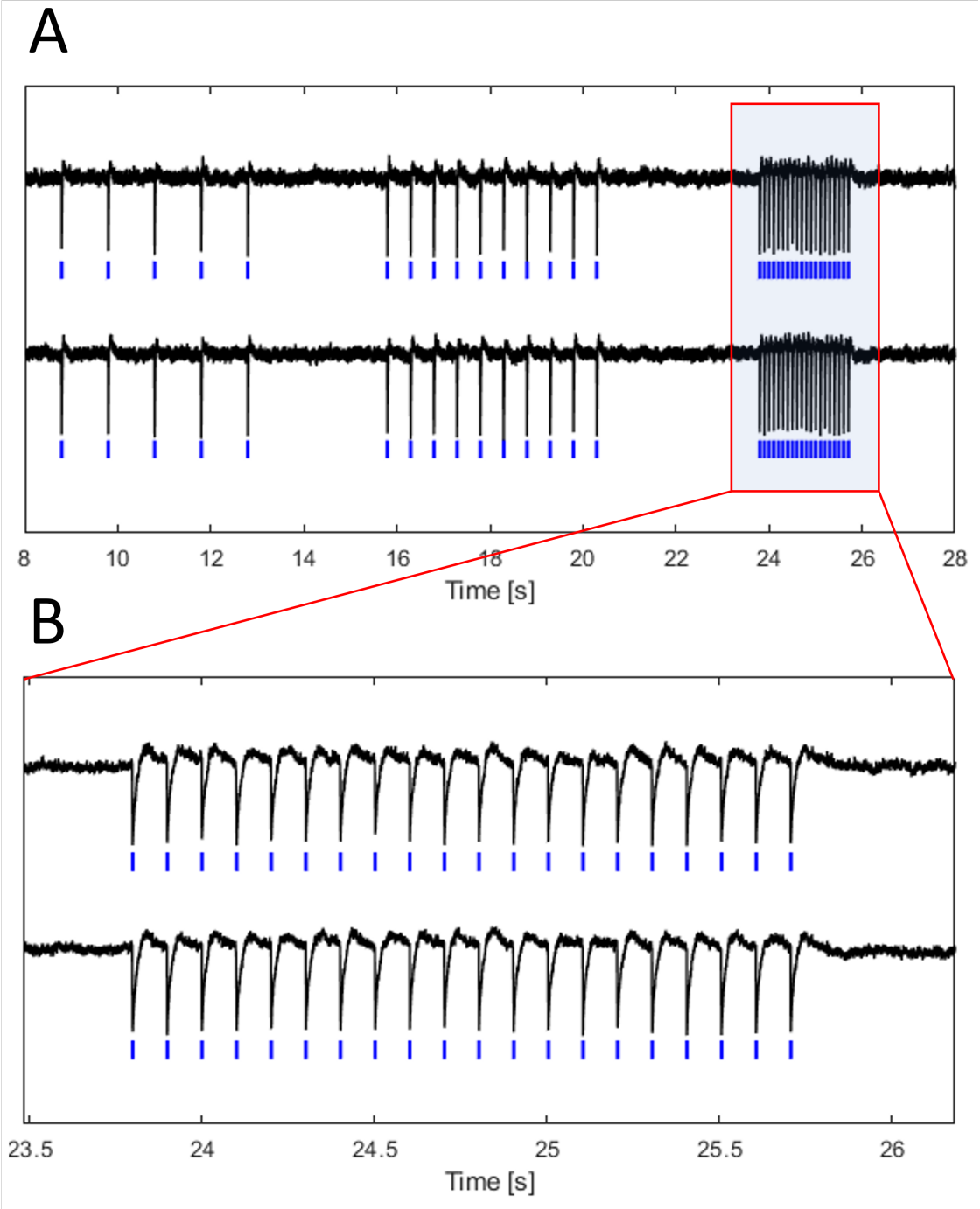

Figura 4.13: validação de *spike detection*

4.4.2.2 Validação de *burst detection* 1

O primeiro teste que realizámos para testar o algoritmo de *burst detection* envolveu a estimulação de um eléctrodo próximo de dois outros que estão a ser monitorizados. Nesta primeira fase não nos preocupámos com a quantidade de *spikes* que continha o *burst*, pelo que definimos como dois o número mínimo de *spikes* para detectarmos um *burst*, com um intervalo máximo entre *spikes* de 0.5 segundos.

Como podemos observar na figura 4.14 criámos três *spikes* de um em um segundo seguidos de dez *spikes* com um espaçamento de cerca de 0,08 segundos. Assim, sabemos previamente que os primeiros não devem constituir um *burst* e os seguintes devem detectá-lo.

Da mesma forma que na validação anterior, fizemos a exportação dos dados e corremos o *script* de *MATLAB* para nos certificarmos que todos todos os eventos forma detectados. A azul, em baixo de cada um dos sinais, podemos observar os *spikes* que correspondem a cada estímulo detectado e a vermelho o inicio de cada *burst*.

4.4.2.3 Validação de *burst detection* 2

Na segunda validação queríamos verificar se apenas seriam considerados como *bursts* os blocos de *spikes* que tivessem pelos menos um número mínimo de *spikes*. Neste caso, predefiniu-se que os *bursts* deveriam ter 5 ou mais *spikes*. Assim, gerámos, de três em três segundos, uma bateria de *spikes* que preenchem o requisito de tempo máximo, incrementando a cada um destes ciclos um novo *spike*. Na figura 4.15 podemos ver que os primeiros que nos primeiros dois ciclos, com 3 e 4 *spikes* por bloco, não foi detectado um *burst* já que o número mínimo de *spikes* estava definido como sendo 5.

4.4.2.4 Validação de *burst detection* 3

Na última validação do algoritmo de *burst detection* decidimos estimular independentemente dois canais, de modo a percebermos se a detecção que estamos a fazer para diferentes canais é independente entre eles. Podemos observar na figura 4.16 que a detecção é feita sem quaisquer problemas e também é interessante ver qual é o efeito que a estimulação num canal provoca no outro que está próximo. Conseguimos, então, reparar que o canal que se encontra em cima, quando é estimulado e apresenta potenciais elevados, influencia o sinal do canal de baixo, gerando algum ruído no mesmo instante em que ocorrem os *spikes*.

4.4.2.5 Validação de *Network Burst*

Na validação do *Network Burst detection* geramos novamente um conjunto de *bursts* independentes para cada canal, definindo o número mínimo de canais em *burst* como 2 para que se detete um *network burst*. O segundo canal apresenta espaçamentos iguais entre todos os *bursts*, já no primeiro suprimimos alguns destes, de modo a verificar se eram detectados *network bursts* apenas

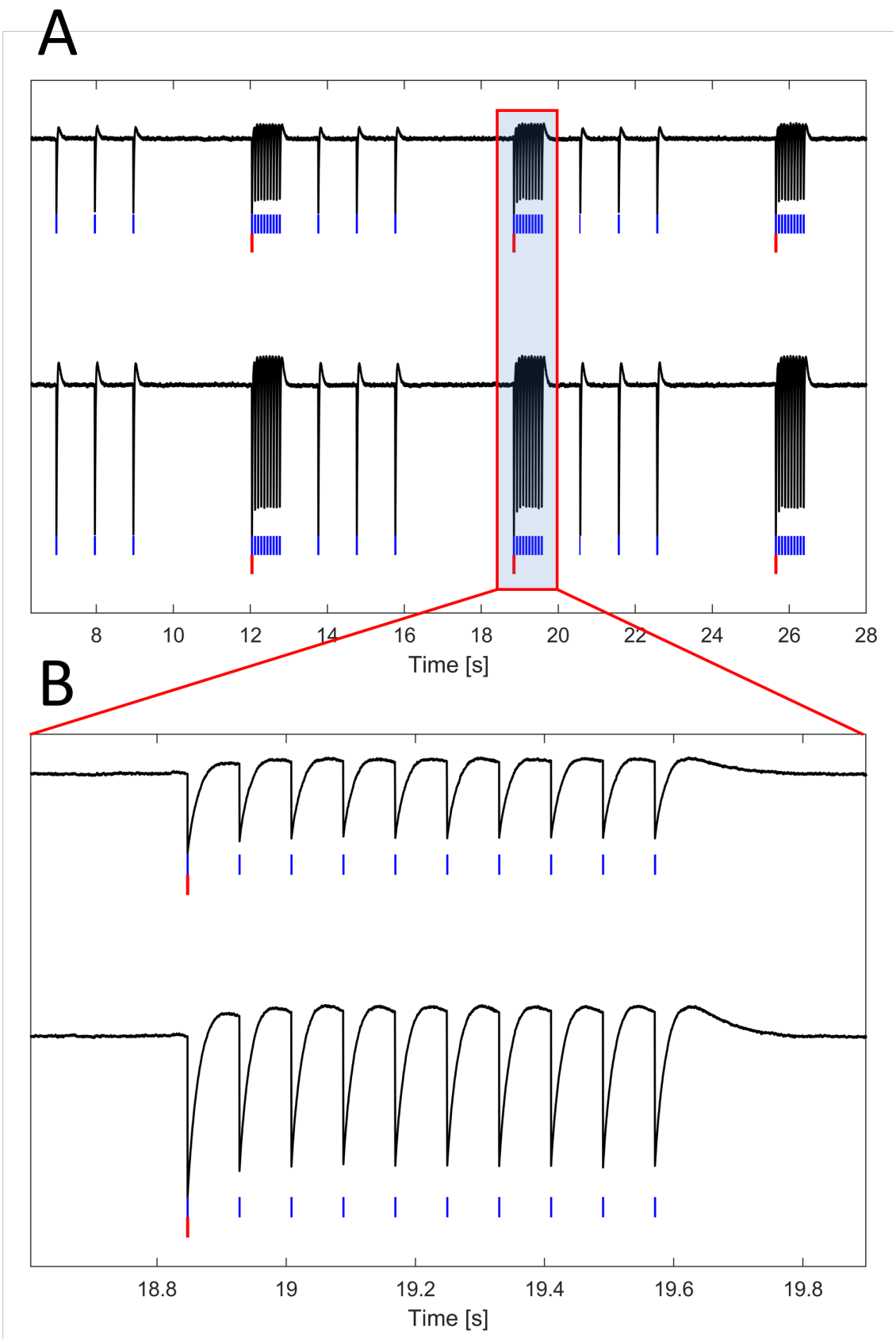

Figura 4.14: validação de *burst detection* com dois canais - tamanho de *burst* 2

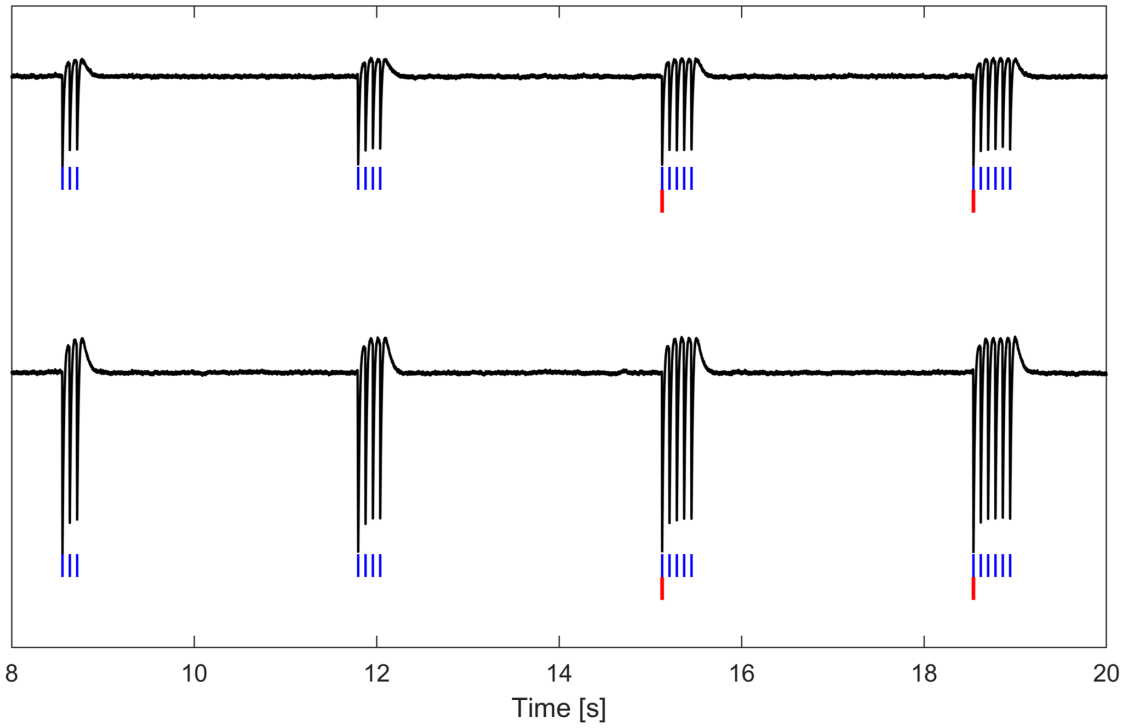

Figura 4.15: validação *burst detection* com dois canais - tamanho de *burst* 5

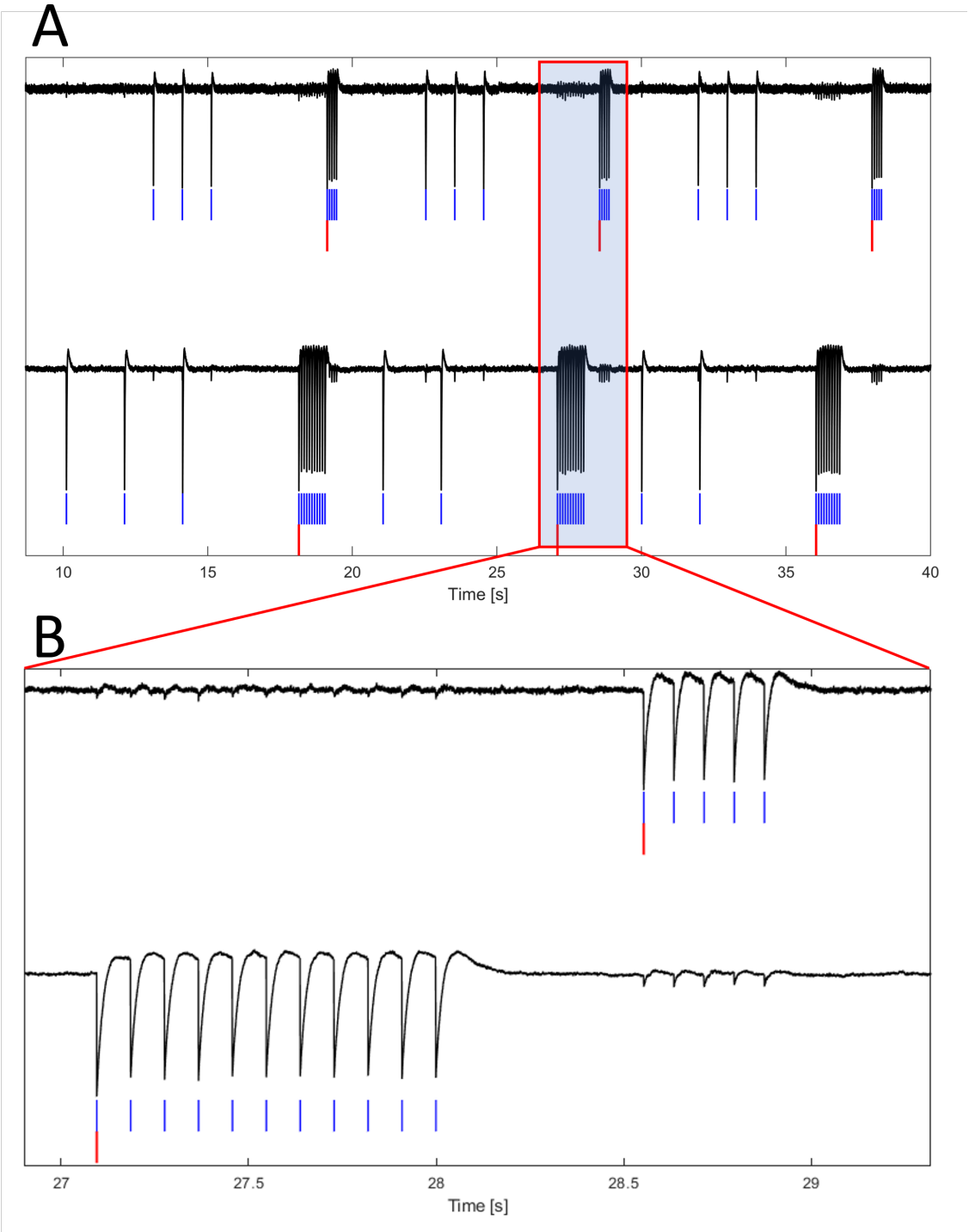

Figura 4.16: validação *burst detection* com dois canais - tamanho de burst 5 - eléctrodos com sinais diferentes

quando ambos os canais se encontram em *burst*. Na figura 4.17 podemos observar a vermelho a detecção dos *bursts* de cada canal e a azul os *network bursts*.

4.5 Conclusões

Após estas validações terem sido realizadas pudemos concluir que estes algoritmos funcionam de forma eficaz. Tendo sido essencial o uso preliminar do sistema de simulação (*LCDK*), já que permitiu um avanço na validação destes algoritmos.

Podemos concluir também que existem várias formas de organizar os nossos *interrupts* de modo a obtermos diferentes performances do nosso sistema.

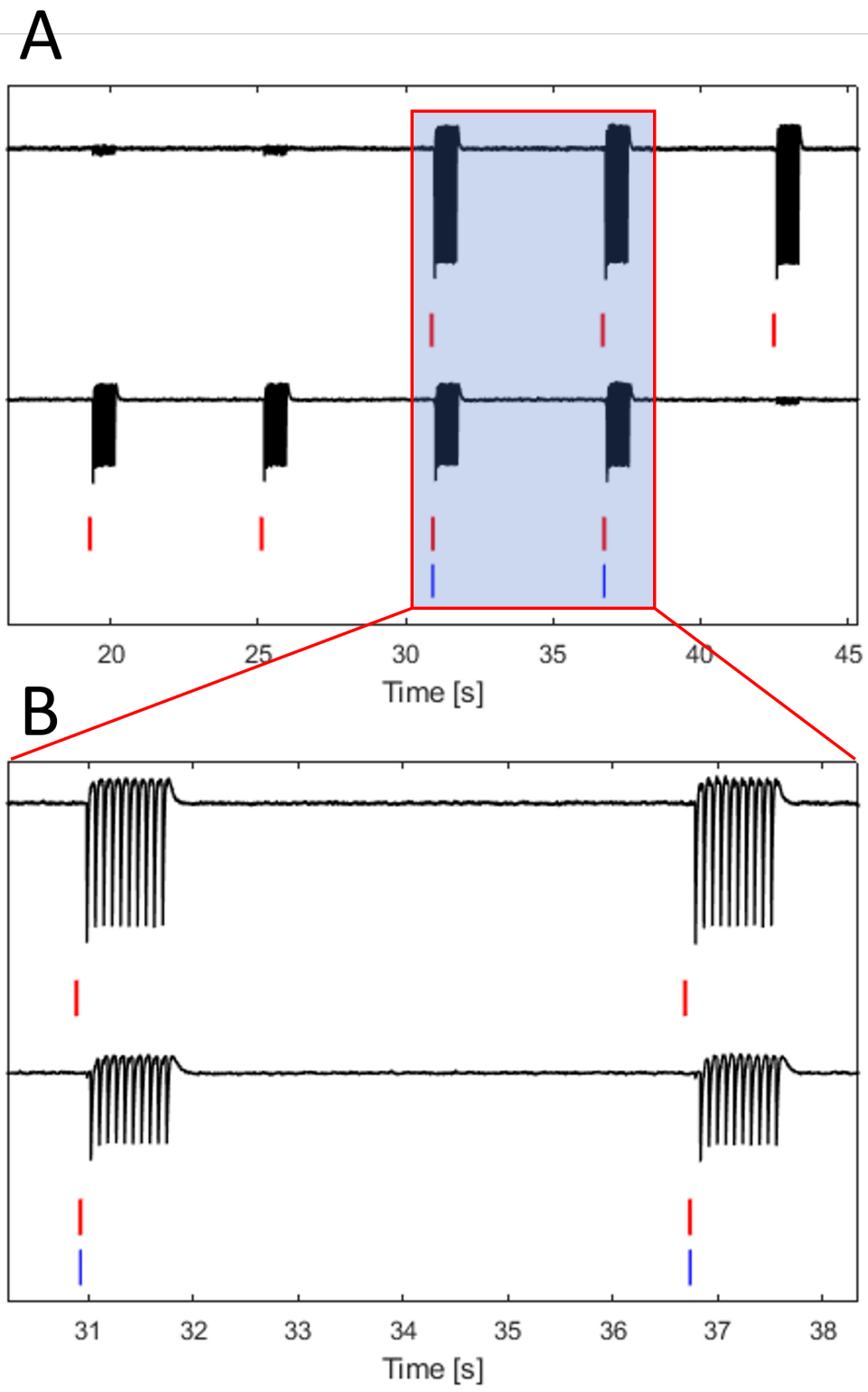

Figura 4.17: validação *network burst* com dois canais

Capítulo 5

User Interface

Para a realização da interface gráfica utilizámos o *Windows Forms*, que é uma *user interface framework*, que nos ajuda bastante a realizar formas básicas, como janelas e botões, poupando-nos algum tempo na implementação. Quando necessitamos implementar funcionalidades gráficas mais complexas o código foi escrito manualmente.

5.1 Menu

Nós temos uma classe principal, à qual demos o nome de *Neural App*, que contém a nossa janela principal e botões que nos dão acesso às funcionalidades da aplicação, às definições dos parâmetros dos algoritmos e escolha de eléctrodos, à conexão entre computador e *DSP* e à visualização de detecções através de diferentes tipos de gráficos. O *layout* desta janela é igual tanto para o sistema de simulação como para o sistema real, na figura 5.1 podemos observar o aspecto da mesma.

Figura 5.1: Janela Principal - Neural App

5.2 Kit

Antes de começarmos a desenvolver e testar código para comunicar com o *MEA2100*, começámos por fazê-lo com o *LCDK*, o que acabou por tornar-se a base do que foi feito posteriormente com o sistema do laboratório. Daí a necessidade de mostrar-mos o que foi feito inicialmente neste contexto.

5.2.1 *Settings*

Quando o utilizador clica no botão com a *label Settings* este abre uma nova janela onde vai ser feita a escolha de canais a monitorizar.

5.2.1.1 Escolha de Eléctrodos

Nesta classe, criaram-se dois métodos para elaborar o *layout* presente na figura 5.2, um que cria os botões que podem ser usados pelo utilizador para escolher eléctrodos individuais e o outro cria as *check boxes* para seleccionar colunas ou linhas de eléctrodos.

Para criar os botões temos que aceder a um ficheiro *electrodelabels.txt* onde estejam presentes os nomes de cada eléctrodo por ordem em que são recebidos no *MeaData*, que vai do G13 ao G15 (o nome dos eléctrodos e a sua ordem não apresentam qualquer relação, daí a necessidade deste método). A informação de cada um dos eléctrodos é guardada numa estrutura que contém a sua *label*, a letra, o número e o **id**, que por sua vez são organizados numa lista, sendo esta posteriormente ordenada por letra e depois por número. Temos ainda uma outra lista de inteiros que apenas guarda o *id* dos eléctrodos que são monitorizados, tornando-se esta a chave para o acesso a esses eléctrodos.

5.2.2 *Connect*

Após a configuração estar concluída o utilizador terá que conectar o computador ao *DSP*, isso é feito clicando neste botão, que cria um objecto *serialCOM*.

O constructor da classe *serialCOM* cria um objecto da classe *SerialPort*, presente no *namespace System.IO.Ports*. Este constructor pode ser chamado no modo *default* em que todos os parâmetros da comunicação via porta série já estão estabelecidos com a*COM4* como a porta associada à comunicação, com velocidade de transmissão de 115200 bps, com 8 *bits* por *frame*, sem *bit* de paridade e um *stop bit*, ou no modo em que o utilizador possa fazer esta configuração como lhe convier. Fizemos também uma série de métodos nesta classe que executam várias formas de envio de dados, também para explorar as potencialidades da comunicação, já que apenas podemos enviar 8 *bits* de cada vez, que corresponde ao tamanho de um *char*. O que quer dizer que todos os dados a serem enviados pelo *DSP* terão este formato, que será posteriormente convertido em valores inteiros por parte do computador. Quando a conexão estiver estabelecida, então o botão *plot* torna-se visível.

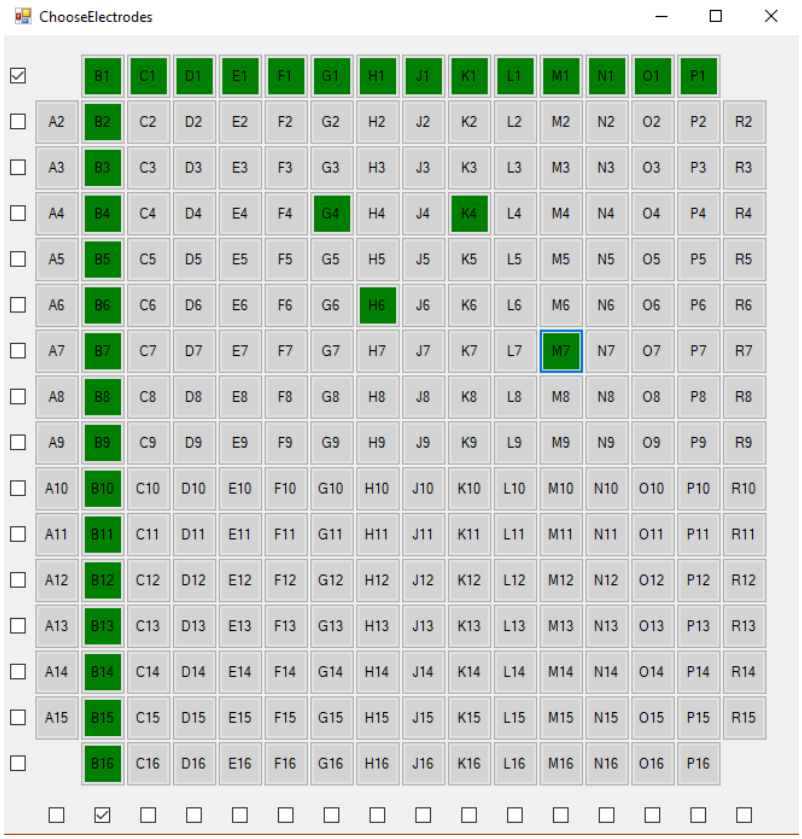

Figura 5.2: Escolha dos Eléctrodos por parte do utilizador

Antes de conectarmos o *DSP* ao *PC* o código do processador deve ser posto a correr manualmente, já que estamos a utilizar o modo *debugging* no *CCS*. Assim, o processador vai esperar que o computador lhe envie informação acerca de que canais irá monitorizar. Quando clicamos no botão *Connect* é então enviado o número de canais a serem monitorizados e a posição a que corresponde cada canal no vector de criação de dados *MeaData*. Depois serão feitas configurações adicionais no *DSP* até o código entrar no ciclo infinito, onde será feita a criação e detecção de eventos.

5.2.3 *Plot*

Este botão só deve ser clicado após a ligação entre *DSP* e *PC* estar feita, daí a necessidade de o tornar apenas visível quando este puder ser utilizado.

Quando o utilizador clicar neste botão, vai ser criado um novo objecto *Graphs*, que abre uma nova janela onde posteriormente vão ser visualizados os gráficos de cada canal (cada um destes gráficos corresponde a um objecto da classe *Chart*). Criámos algumas formas diferentes de visualização dos canais, para percebermos quais poderiam ser as mais interessantes. Uma das formas utilizadas foi mostrar apenas os canais que são monitorizados por ordem de *IDs*, o que pode ser interessante para podermos colocar os gráficos de cada canal o maior possível. Outra forma interessante é colocar os canais como estão distribuídos no *MEA*, com as colunas identificadas de "A"a "R"e as linhas de "1"a "16", assim sabemos exactamente qual é a posição do eléctrodo que estamos a analisar.

Estes gráficos podem mostrar-nos diferentes coisas consoante a nossa necessidade. Podemos ver os dados que vão sendo criados para cada canal, o que na prática terá pouco uso, podemos ver os *spikes* a serem detectados, nos canais a serem monitorizados, ou ver os *interspike* ou *interburst histograms*, que no caso da simulação se torna bastante mais interessante que os outros. No sistema real, porém, iremos ter ainda outro modo de visualização que ainda iremos discutir.

5.2.3.1 *Spike Detection*

Neste modo de visualização conseguimos ver, para cada um dos canais seleccionados, os *spikes* que vão sendo detectados pelo *DSP* ao longo do tempo, sendo que a janela temporal é reduzida devido ao elevado número de gráficos que temos que visualizar. Na figura 5.3 podemos ver que neste modo de visualização mostramos todos os gráficos de cada canal monitorizado seguidos, em vez de aparecerem na posição que equivale à do *MEA*.

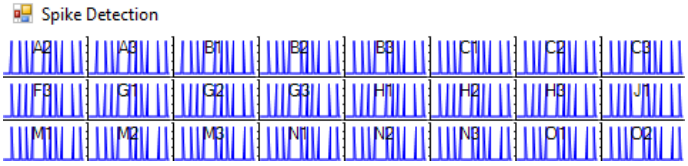

Figura 5.3: Visulização de *Spike detection*

5.2.3.2 *interspike histogram*

O *interspike histogram* é um histograma que nos mostra quantos *spikes* ocorrem dentro de certos intervalos de tempo. Ao criarmos o método decidimos dar a liberdade ao utilizador de escolher qual o tamanho desses intervalos e posteriormente analisar os resultados. É, então, calculado quanto tempo passa entre *spikes* consecutivos, e é incrementado o número de *spikes* na respectiva janela de intervalo de tempo.

Este algoritmo foi realizado apenas para o computador, correndo assim em *C#*, já que o seu resultado serve como informação para realizar análise estatística e seria um desperdício de recursos do *DSP*. Podemos observar estes dados através da interface gráfica à medida que a informação dos *spikes* é recebida pelo computador.

Na figura 5.4 temos exposto um dos modos de visualização do *interspike histogram*, que consiste na visualização ordenada dos gráficos por ordem de *IDs* dos *electrode labels*. Foi cortada parte da janela da figura por uma questão espaço, já que o espaço dos gráficos do eléctrodos que não são lidos não contêm qualquer informação.

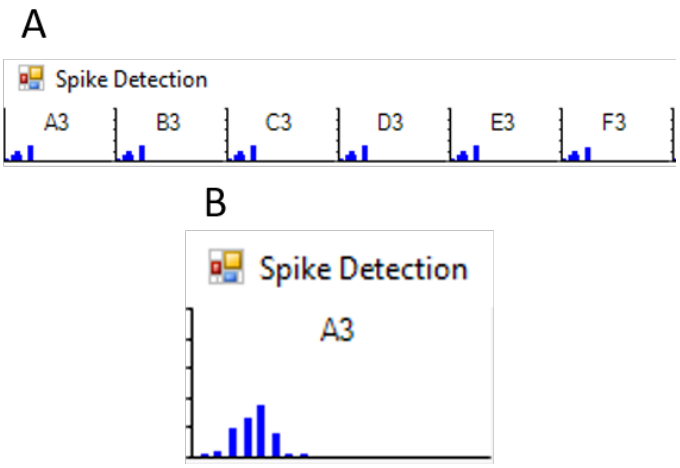

Figura 5.4: Visulização de *Interspike histograms*

5.3 MEA

5.3.1 *Settings*

Quando o utilizador clica no botão com a *label Settings* é estabelecida a comunicação entre o computador e o *MEA2100* e é criado um novo objecto *Settings*, abrindo assim uma nova janela onde vai ser feita a configuração da monitorização (figura 5.5).

Antes da configuração ser feita fazemos a conexão entre os dispositivos através da classe *CMcsUsbNet* presente na *DLL* da *MCS*, com a qual conseguimos saber que dispositivos da *MCS* estão a ser detectados pelo computador, qual a entrada do *USB* do *MEA2100*, etc. Com essa

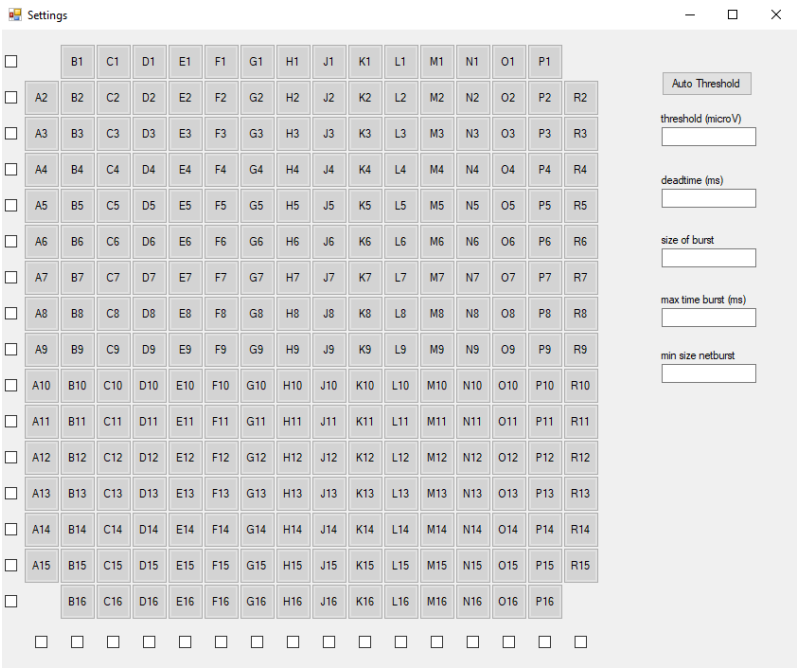

Figura 5.5: Janela de *Settings* do *MEA*

informação podemos então conectar o *PC* ao *MEA2100* (o computador não acede directamente à *DSP* presente na *interface board*, mas sim a dispositivos que partilham informação com ambos).

O utilizador pode escolher que canais pretende monitorizar e todos os parâmetros para a detecção de eventos, tais como o *threshold*, *deadtime*, intervalo entre *spikes* para que seja detectado um *burst* e o número de *spikes* mínimo para que este ocorra, bem como o número de canais em *burst* para que ocorra um *network burst*. É também possível que a escolha do *threshold* seja feita automaticamente para cada um dos eléctrodos. Todos estes parâmetros são enviados para a *mailbox* antes que o o código no *DSP* comece a correr e são lidos por ele através de uma função criada para receber dados da *mailbox* antes que os *interrupts* estejam activos.

5.3.1.1 *Auto Threshold*

Até agora apenas fizemos testes em que estimulámos os eléctrodos do *MEA*, por isso podíamos utilizar um *threshold* fixo para todos eles. No entanto, quando no *MEA* se encontram células vivas vemos que cada eléctrodo terá um ganho de ruído diferente e, para além disso, a tensão dos potenciais de acção é bem menor no caso das células do que as que temos usado nos testes, daí a necessidade de encontrarmos uma forma de definirmos um *threshold* diferente para cada eléctrodo consoante a necessidade de cada um.

Para isso criamos uma nova classe que possa fazer esta tarefa. Será criado um objecto desta classe assim que o botão com a *label Auto Threshold* for clicado. Tivemos que usar as funcionalidades da *DLL* disponibilizada pela *MCS* para criarmos esta classe, que nos permitiu gravar dados directamente vindos do *MEA2100* no computador. De um modo geral o que esta faz é guardar

um *sweep* de dados de cada eléctrodo do *MEA* durante um segundo. Depois calcula a média da tensão em cada eléctrodo e posteriormente o seu desvio padrão. O utilizador previamente escolhe quantas vezes o *threshold* de cada eléctrodo será maior que o desvio padrão. Enquanto isto ocorre podemos ver, ao clicarmos no botão *StartDAC*, o *sweep* de dados de cada eléctrodo numa matriz de gráficos, e passado o segundo de aquisição então os *thresholds* aparecem para cada eléctrodo, como podemos ver na figura 5.6. Se os *thresholds* não estiverem como pretendido basta repetir a aquisição.

Quando fechamos esta janela é chamado um evento associado a esta acção, na qual a sua função associada envia os *thresholds* de cada eléctrodo de forma idêntica à que é feita com a escolha dos eléctrodos, através da *mailbox*, a partir da posição 0x400 em diante. Na figura 5.6 podemos ver como visualizamos o modo de *auto threshold*, conseguindo ver as linhas de *threshold* em relação ao sinal medido na janela de 1 segundo.

Figura 5.6: Janela de *Auto Threshold*

5.3.2 *Connect*

Neste caso, o botão *Connect* serve para fazermos o *load* do código compilado em modo *realise*. Para isso temos que utilizar uma classe presente no *DLL*, *CMcsUsbFactoryNet*, na qual se encontra o método que nos permite enviar o ficheiro com extensão *.bin* para o *DSP*.

Para obtermos este ficheiro binário temos que fazer um *postbuild batch script* de modo a convertermos o ficheiro *.out*, obtido após a compilação, num ficheiro binário.

5.3.3 *Plot*

Quando o utilizador clicar neste botão, vai ser criado um novo objecto *Graphs*, que abre uma nova janela onde posteriormente vão ser visualizados os gráficos. Existem três tipos de gráficos possíveis de serem visualizados, sendo que apenas os dois primeiros foram testados, o *Spike Detect*, o *Interspike Histogram* e o *Interburst Histogram*.

O utilizador ao clicar no botão *Plot* tem duas opões de visualização. Poderá ver um *raster plot* dos eventos que vão sendo recolhidos através do *DSP*, sendo que os pontos azuis representam os *spikes*, os vermelhas os *bursts* e os losangos roxos representam os *network bursts*. As ordenadas do gráfico apresentam o número dos canais e as abcissas os instantes de tempo a que ocorrem os eventos em segundos, sendo que estes flutuam numa janela de 25 segundos que avança de 2 em 2 segundos.

A figura 5.7 mostra-nos o *raster plot* apenas com *spikes*, já que estes apenas estão a ocorrer de segundo a segundo.

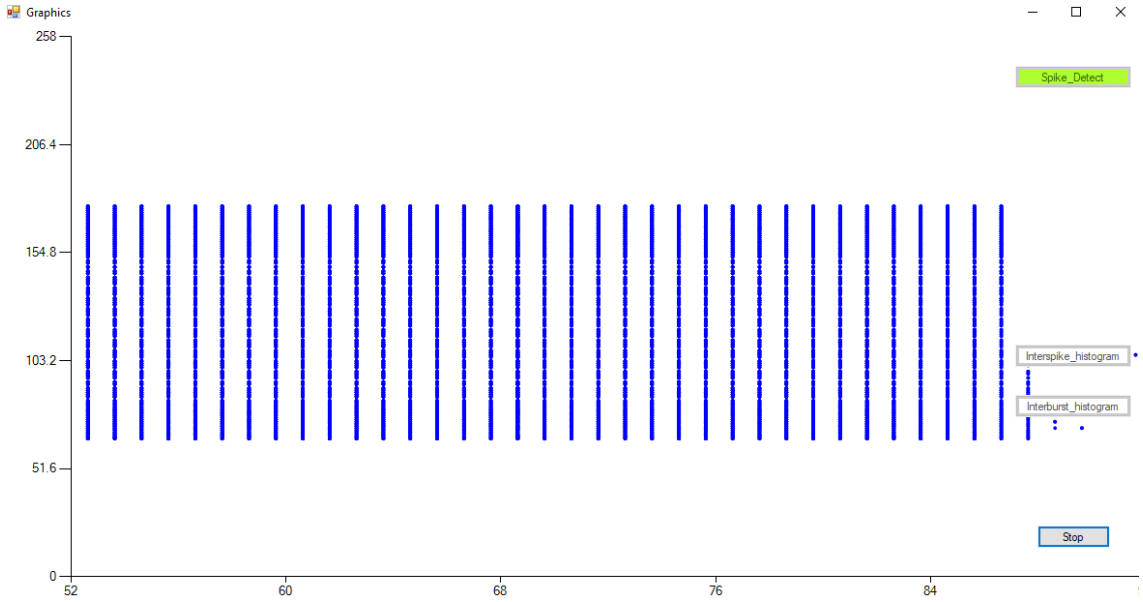

Figura 5.7: Visulização de *raster plot* para 96 eléctrodos com *spikes* a ocorrerem a 1 Hz

Na figura 5.8 podemos ver como são visualizados os *interspike histograms* na nossa *user interface*. Esta imagem foi captada a quando de uma monitorização com células, sendo que estas estas estão dividas em seis zonas e estávamos a captar apenas uma delas (42 eléctrodos), daí apenas serem visíveis esses eléctrodos.

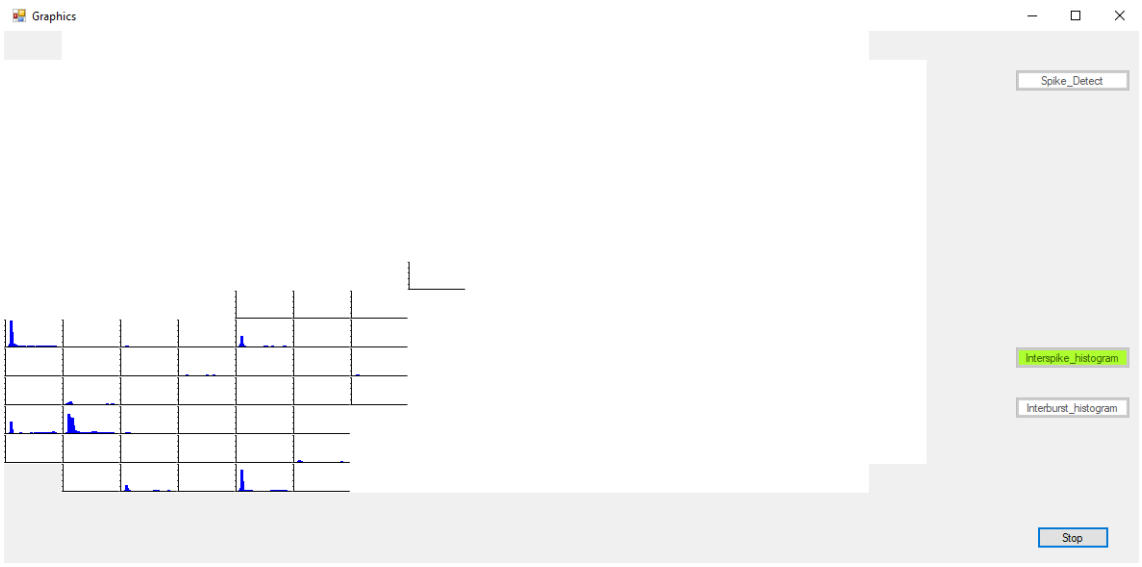

Figura 5.8: Visulização de *interspike histogram* de um poço de *6-well-MEA*

Capítulo 6

Comunicação

Neste capítulo abordamos em detalhe como foi feita a comunicação entre o *DSP* e o computador.

Começamos por descrever como é feita a comunicação entre o *Kit*, utilizado para simular o nosso sistema, e o *PC*, e em seguida descrevemos detalhadamente como é feita a comunicação entre o *DSP* presente no *MEA2100-256* e o computador.

6.1 Kit

6.1.1 Comunicação *DSP/PC* - UART

A comunicação entre o processador c6748 e o *PC* foi feita de forma distinta da comunicação entre *MEA2100* e *PC*. Inicialmente começámos por escrever código para o *LCDK*, que contém uma porta *UART* usada para comunicar com o computador. Pensámos que o sistema do laboratório também tivesse este sistema incluído, no entanto quando começámos a testar código no *MEA2100* apercebemo-nos que este não possuía um dispositivo *UART*, em vez disso pode utilizar o protocolo i2c, que, por sua vez, é chamado por um *interrupt* do *DMA*, ou então a *mailbox*, que iremos aprofundar mais à frente.

UART é o acrónimo para *Universal Asynchronous Receiver/Transmitter*. Não é um protocolo de comunicação como *SPI* ou *I2C*, mas sim um circuito físico num micro-controlador ou circuito integrado. O seu objectivo é receber e transmitir dados, usando apenas dois cabos para transmitir informação entre dispositivos. Dois *UARTs* comunicam directamente um com o outro, o *UART* de transmissão converte dados em paralelo de um dispositivo de controlo (pe. CPU) para dados em série, transmite-os para o *UART* de recepção, que depois reconverte os dados para paralelo. Os dados são transmitidos do pino *Tx* do transmissor para o pino *Rx* do receptor de forma assíncrona, o que significa que não existe sinal de *clock* para sincronizar o output dos bits transferidos do *Tx* e o *sampling* dos bits pelo *Rx*. Em vez de sinal de *clock*, utiliza *start*, *stop* e *parity* (opcionais) bits nos dados a serem transferidos. Estes bits definem o início e o fim de pacotes de dados para que o *UART* receptor saiba quando começar a ler os dados. Quando o *UART* receptor detecta o *start bit* , começa a ler os *bits* a uma frequência específica, conhecida como *BAUD rate* (medida

de velocidade de transferência de dados, expressa em bits/s). Ambos os *UART* devem trabalhar no mesmo *BAUD rate*.

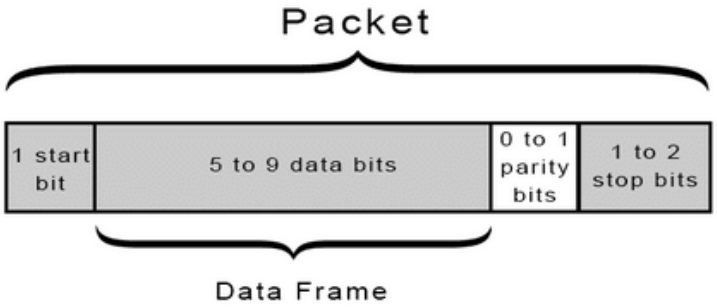

Figura 6.1: Comunicação UART

A recepção de eventos por parte do *DSP* é feita quando clicamos no botão *Start* na janela *Plot* da *user interface*. Antes de clicarmos nesse botão devemos decidir quais são os dados que queremos visualizar.

Assim que o botão *Start* é clicado é criada uma *thread* que irá correr o código de recepção de dados num ciclo infinito, até que queiramos parar a gravação. Esta *thread* corre em paralelo com a que trata de mostrar os gráficos.

O que esta *thread* faz é esperar que sejam recebidos novos dados através dos métodos da classe *SerialCOM*. Assim que o *DSP* começa a escrever o computador começa a ler esses dados. O computador irá receber o número do canal a que se refere uma certa detecção, que será guardado localmente, e escreve imediatamente o número do canal no ficheiro de detecção dos *spikes*, depois deverá receber o *timestamp* em que ocorreu a detecção e guardá-lo na posição do vector que corresponde à estrutura daquele canal e mais uma vez escrever no ficheiro este valor. No caso de termos escolhido a visualização do *interspike histogram* deverá ainda correr o algoritmo que o gera.

Com esta informação podemos concluir que os dados ficam guardados nos ficheiros com o número do canal seguido do instante em que ocorreu um evento. É essencial gravação destes dados, quer para fazer *debugging* e validação do código como para posterior análise dos dados por parte dos investigadores.

6.2 MEA

6.2.1 Comunicação *DSP/PC*

Para realizar o envio de dados entre o *DSP* e o *PC* analisámos alguns códigos de exemplo fornecidos pela *MCS* e chegámos à conclusão que a melhor forma de fazermos a comunicação seria através do uso do protocolo *i2c*, já que encontrámos alguns exemplos de envio de parâmetros para o *DSP*, através do *PC*, utilizando este protocolo que fora implementado pela empresa. Do

lado do computador, a necessitamos de utilizar a *DLL* da *MCS* que contém certas classes que nos permitem comunicar com a *interface board*, que por sua vez comunica com o *DSP*, no entanto, quando tentámos utilizar esta ferramenta para comunicar com o *DSP* apercebemo-nos que não éramos capazes de realizar a tarefa requerida. Tivemos dificuldades em perceber se a causa era o facto de estarmos a utilizar a classe errada, devido à falta de documentação é-nos difícil saber, ou simplesmente porque esta não podia ser utilizada neste sistema, portanto tivemos que entrar em contacto com os engenheiros da *MCS* para nos esclarecerem esta dúvida, ao que nos responderam que não deveríamos utilizar este protocolo para a comunicação e que seria melhor se utilizássemos a *mailbox* para comunicarmos entre dispositivos, o que acabamos por fazer.

A informação tanto pode ser enviada do *DSP* para o *PC* como no sentido inverso. Enviamos informação do computador para o *DSP* apenas para definirmos parâmetros a serem utilizados nos algoritmos de detecção de eventos e para sabermos que canais irão ser monitorizados, o que é feito anteriormente ao início da detecção em si. Enviamos também uma *flag* que informa o *DSP* se o computador já recebeu os últimos dados enviados por este, para que possa enviar uma nova remessa sem por em causa a actual. No caso do envio de informação do *DSP* para o computador, enviamos periodicamente dados relativamente à detecção de eventos. Isto é feito a cadências definidas no código *C#*, sendo que a cada evento corresponde uma cadência diferente. Os dados enviados são relativos aos tempos em que ocorreram os *spikes*, *bursts* e *network bursts* de cada canal a ser monitorizado, que são gravados posteriormente em ficheiros de texto e visualizados em tempo real na *user interface*.

6.2.1.1 *Mailbox*

O espaço reservado para a *mailbox* encontra-se na região de memória, mapeada no *DSP*, entre 0XA0001000 e 0XA0001FFC, o que corresponde a 4092 posições de memória, no entanto uma posição de memória só contém 8 *bits* (1 *byte*), pelo que temos que agregar quatro destas posições para obtermos os 32 bits necessários para o envio dos dados que pretendemos. Assim sendo, em vez das 4092 posições de memória temos na realidade apenas 1023, com esta redução na memória disponível tivemos que arranjar uma forma segura de enviarmos os dados faseadamente no caso do espaço não ser suficiente. Para além disso, apenas podemos escrever dados do tipo *unsigned int* nestas posições de memória, o que nos obriga a realizar uma conversão dos dados que iremos enviar, pois esses dados são os tempos em que ocorreram eventos.

6.2.1.2 *Interrupt mailbox*

Foi criado um *interrupt* que é chamado assim que o computador escreve para um endereço associado à *mailbox*, este *interrupt* é activo através da mudança de estado de um *gpio* que ocorre porque existe um pino (número 6) que está associado à escrita por parte do computador na *mailbox*.

Nesse *interrupt* existe uma instrução *switch* que verifica em que endereço de memória escreveu o computador, consoante esse endereço de memória são realizadas diferentes tarefas. Isto torna-se possível já que existe um outro endereço específico (0xb28) que nos diz qual foi o ultimo endereço

a ser escrito na região de memória da *mailbox*. Os primeiros *cases* presentes no *switch* estão associados aos primeiros endereços de memória da *mailbox*, que reservamos para a modificação de parâmetros das funções de detecção de eventos, caso seja necessário alterá-los com estas já a correr. Na figura 6.4 vemos a estrutura deste *switch*.

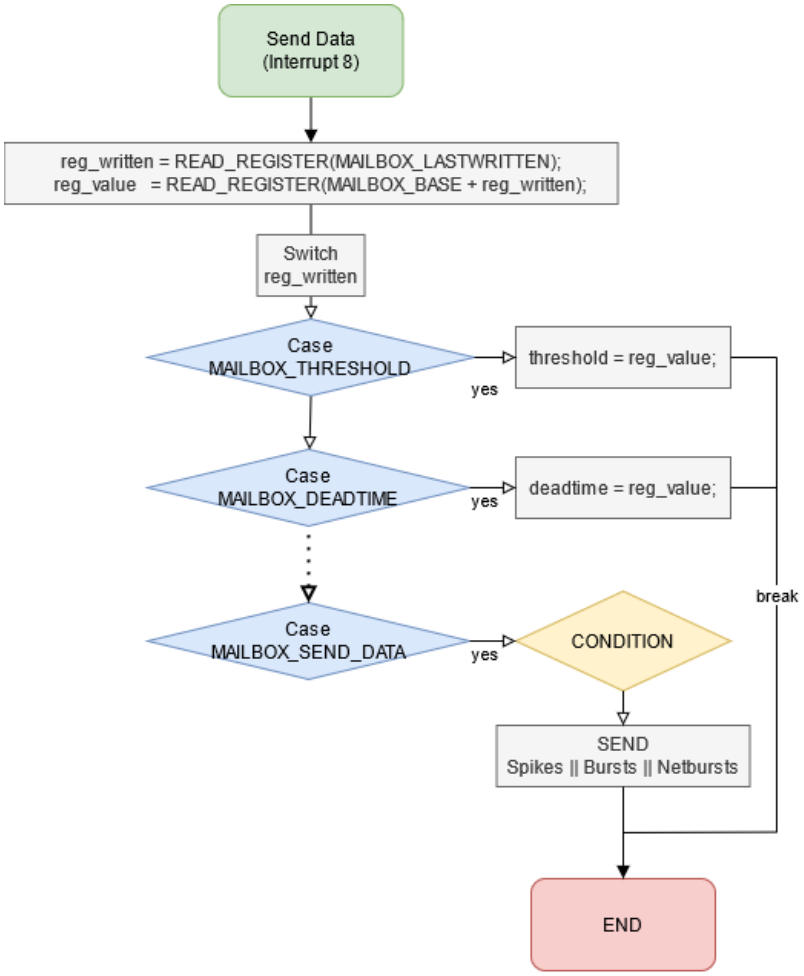

Figura 6.2: *Mailbox interrupt 8*

6.2.1.3 *Polling*

A forma que achamos mais conveniente para fazermos o envio dos eventos por parte do *DSP* foi através de *polling* de dados feito por parte do computador. Este é um processo que consiste no pedido de dados por parte do computador em intervalos de tempo definidos. Existem, então, três temporizadores, um para cada evento, programados no computador, na aplicação *C#*, que quando chegam ao fim de um certo espaço de tempo geram um evento que chama uma *callback function*, fazendo o pedido de recepção de novos dados do evento em causa ao *DSP*. Poderíamos eventualmente enviar a informação destes eventos sempre que recebêssemos uma nova detecção,

assim como é feito na simulação, no entanto isto iria ocupar tempo precioso de processamento ao *DSP*, pelo que decidimos enviar um conjunto maior de dados de uma vez só.

Como já foi referido na secção 4.1, os vectores que guardam os *timestamps* em que ocorreram os eventos para cada canal têm um espaço limitado, portanto é importante que estes dados sejam enviados para o computador antes que já não estejam disponíveis, daí definirmos intervalos de tempo que nos assegurem que esses dados não serão perdidos. Definimos então que o computador pedirá novos dados de *spikes* de 10 em 10 ms, de 1 em 1 segundo de *bursts* e de 10 em 10 segundos de *network bursts*. O que quer dizer que teoricamente apenas poderemos ter, no máximo, 50 *spikes* a cada 10 ms para que os dados não sejam perdidos por parte do computador. Na figura 6.3 podemos ver como e a que cadências são feitos os pedidos de dados por parte do computador.

Inicialmente testámos este envio de dados separadamente, começando a fazer o envio da informação dos *spikes* apenas de dois canais. Conseguimos receber todos os *spikes* neste caso, no entanto, quando juntámos os outros eventos começámos a obter erros em alguns instantes comparativamente aos estímulos que estávamos a gerar através do *Experimenter*. Chegamos à conclusão que o erro estava a ocorrer do lado do *C#*, devido a um problema de concorrência, já que ao utilizarmos três temporizadores diferentes, um para cada tipo de evento (*spikes*, *bursts* e *network bursts*), por vezes estes sobrepunham-se. Ou seja, estávamos a receber dados relativos aos *spikes*, por exemplo, e o temporizador dos *bursts* chegava ao fim e fazia um novo pedido de dados, estes relativos aos *bursts*, causando uma chamada do *interrupt* que ainda estava a enviar os dados relativos aos *spikes*, fazendo com que existisse perda de informação. Para resolvermos este problema pensámos em duas estratégias diferentes, uma que consistia na criação de um semáforo de *busy-waiting* que, no caso de um dos temporizadores ter chegado ao final e a sua *callback function* estar a ser executada, então as as outras teriam que esperar que esta terminasse para executarem a sua tarefa. A outra consiste num *lock* de exclusão mútua para um dado objecto que, como o nome indica, bloqueia um dado objecto partilhado entre estas *callback functions* e que, ao terminar o bloco de código que se encontra dentro dessa instrução *lock*, o liberta e executa o bloco que possa estar pendente pendente.

O método *busy-waiting* é mais dispendioso para o computador, já que este fica "preso"num ciclo infinito à espera que o estado do semáforo se altere, enquanto o *lock* é mais eficiente, logo foi o que acabou por ser utilizado.

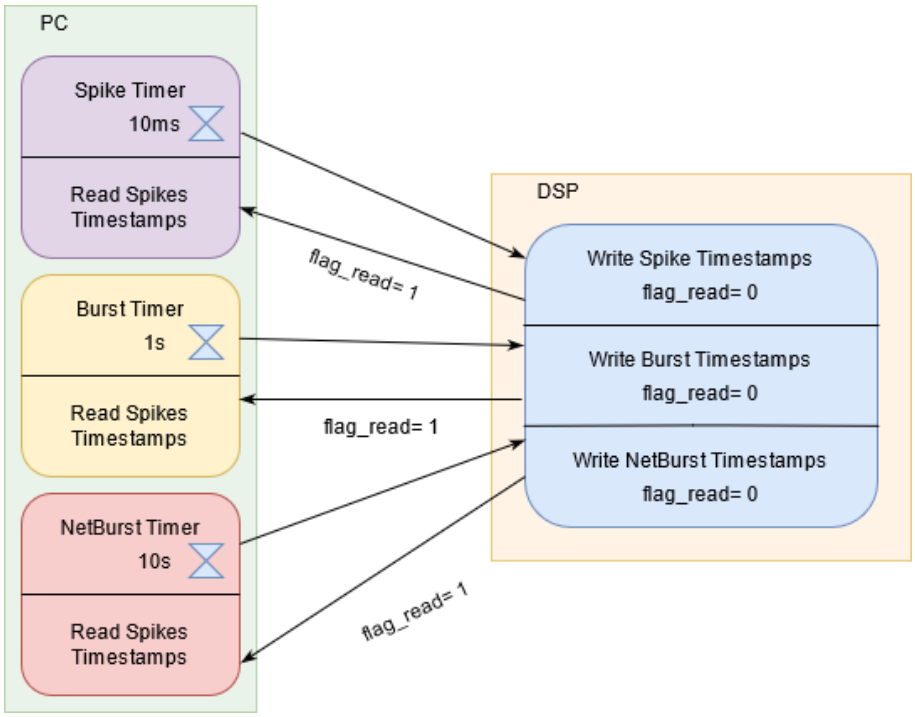

Figura 6.3: Temporizadores - *Polling*

6.2.1.4 Envio de dados

O envio de dados para os diferentes tipos de eventos é feito separadamente, o computador pede ao *DSP* que lhe sejam transmitidos os dados relativos aos *spikes*, ou aos *bursts*, ou *network bursts*. Sendo que o seu formato é semelhante, havendo apenas a diferença de, no caso dos *network bursts*, apenas serem enviados os tempos relativos às detecções sem referência a que canais estão em causa.

Quando o computador faz um novo pedido de dados, é então gerado um *interrupt*, que verifica qual é o evento que deve ser enviado, então começa a executar o código de envio. Inicialmente começa por colocar um registo que definimos a zero, para indicar que o computador deve esperar que o *DSP* escreva os dados na *mailbox*. Então, entra num ciclo que irá percorrer todos os canais que estão a ser monitorizados e no inicio de cada iteração escreve qual o número do canal a que se vão referir os instantes de tempo que se encontrarão a seguir. Sendo que de cada vez que escreve num registo da *mailbox* incrementa um contador que o informa em que posição deve escrever a seguir. Agora entramos no envio dos dados propriamente ditos. Temos duas variáveis auxiliares que nos informam, para o canal em questão, qual foi a última posição lida no vector de *timestamps* e qual foi a posição em que foi escrito o instante da última detecção do evento em causa. Uma condição verifica se a última posição lida é menor que a posição em que foi escrita a última detecção, isto para sabermos se o tamanho do vector já foi excedido, e portanto se há dados que estão a ser escritos no fim do vector e outros que estão a começar a ser escritos no início.

Em caso afirmativo o número de novos *spikes* a serem enviados será calculado pela subtracção destas duas variáveis, caso contrário temos que somar a este valor o tamanho do vector. Após esta verificação estar feita, o *DSP* escreve num registo qual é o número de detecções a serem lidas pelo computador, referentes a um canal. Em seguida temos mais um ciclo cujo o número de iterações corresponde ao número de detecções feitas, que escreve em que instantes ocorreram as detecções, sendo que temos que enviar estes dados com o tipo *uint*, logo dividimos o valor do tempo pelo período de aquisição dos dados, para obtermos um valor que corresponde ao número de contagens feitas pelo relógio, que depois é reconvertido no computador. No final de tudo isto, quando este processo se repetiu para cada canal, coloca o valor do registo que informa o computador se já tem novos dados a 1, e aí o computador lê a informação.

Na figura 6.4 apresentamos um fluxograma que mostra como é o envio dos dados dos *spikes* em específico para uma melhor compreensão do leitor.

Não podemos deixar de referir que não representamos um pormenor deste algoritmo, que não deixa de ser bastante importante. No caso de termos mais dados a enviar do que aqueles que a *mailbox* permite guardar, o *DSP* deve escrever até que esse espaço chegue ao final, esperar que o computador leia os dados e voltar a escrever o dados em falta a partir da posição inicial de escrita que será lida novamente pelo computador.

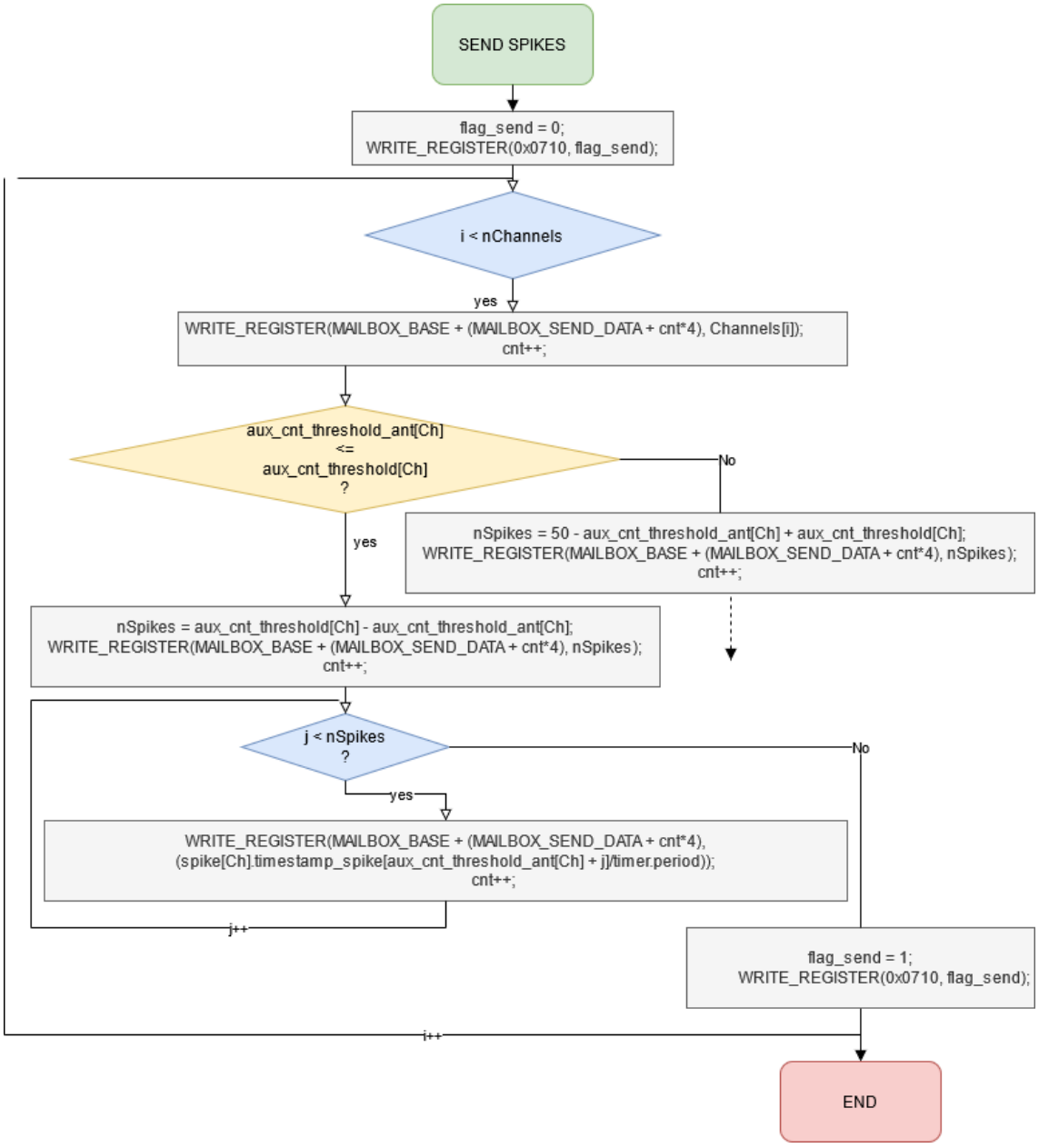

Figura 6.4: Envio de tempos em que ocorreram *spikes*

Capítulo 7

Resultados

7.1 Validação do Sistema *DSP*-Comunicação-*PC*

Esta secção descreve como foram feitas as validações do sistema completo, com a detecção de eventos por parte do *DSP* a correr em paralelo com o envio de dados entre o processador e o computador.

Para fazermos as validações, mais uma vez, fizemos um *script* em *MATLAB* que compara os dados que gravamos no *Experimenter*, *spikes*, com os que foram recolhidos pelo *DSP* e enviados para o computador, sendo estes gravados em ficheiros *.txt*.

O método utilizado para as diferentes validações consiste no incremento do número de eléctrodos a serem monitorizados à medida que cada validação é concluída, na utilização de diferentes períodos de actualização do relógio e na alteração dos períodos de tempo a que são pedidas novas amostras de dados por parte do computador. Para isso estimulámos vários eléctrodos de modo a obtermos conjuntos de dez *spikes* durante um segundo separados de três segundos cada. Os *thresholds* associados à detenção dos *spikes* foram colocados a 1500 µV tanto no *Experimenter* como no *DSP*, o tamanho mínimo dos *bursts* foi definido como quatro e o número mínimo de canais em *burst* para detectarmos *network bursts* a dez. Nas figuras que irão ser apresentadas as circunferências a verde correspondem aos *spikes* capturados no *Experimenter*, já os pontos a preto, a vermelho e a azul correspondem aos *spikes*, *bursts* e *network bursts* capturados pelo *DSP*.

É importante ter em mente que a alteração da frequência de actualização do relógio não prejudica directamente a taxa de amostragem do sinal dos eléctrodos, apenas a precisão do instante de tempo atribuído à detecção do evento. No entanto veremos mais à frente como um aumento desta frequência pode gerar problemas.

7.1.1 Validação com 32 eléctrodos - relógio actualizado a 1000Hz

Começámos por utilizar uma frequência de actualização do relógio de 1000 Hz, com um período de pedidos de *spikes*, *bursts* e *network bursts* de 10, 1000, 2000 ms respectivamente, por parte do computador, monitorizando 32 eléctrodos.

Na figura 7.1 podemos ver que todos os *spikes* que foram exportados do *Experimenter* (representados com circunferências verdes) estão alinhados com os que foram exportados da gravação do *DSP* (com os *spikes* representados com pontos pretos, os *bursts* com pontos vermelhos e os *network bursts* com pontos a azul). Podemos também observar que o *DSP* detectou correctamente todos os *bursts*, sendo que também detectou os *network bursts* não estando eles representados na imagem. O eixo das abcissas representa o número do canal, que é representado pela ordem em que é recebido pelo *DSP*. O eixo das ordenadas representa o tempo em segundos.

Na imagem *A*, onde se vêem todos os eléctrodos que foram monitorizados quer no *DSP* como no *Experimenter*, podemos ver a gravação que foi feita de apenas 30 segundos. Verifica-se que existem eléctrodos a mais na gravação do *Experimenter*, que correspondem às circunferências a verde, devido a facto de se ter monitorizado mais do que 32 canais nesta gravação.

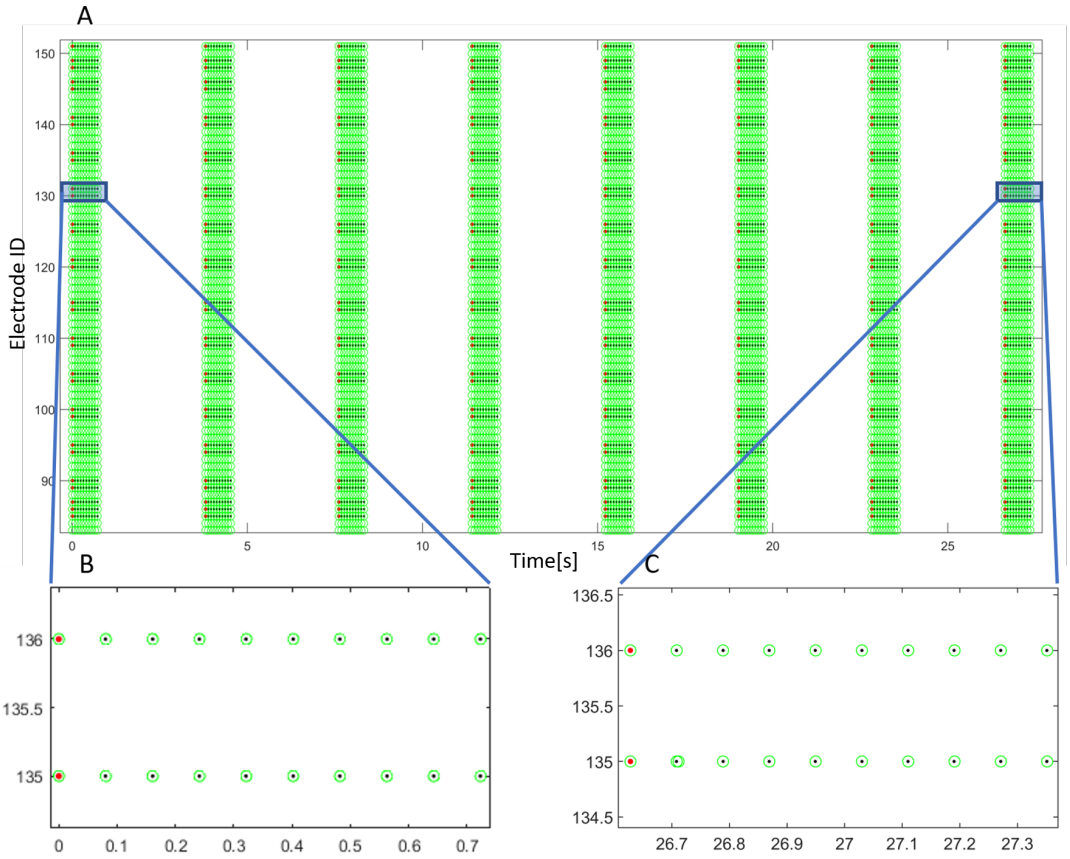

Figura 7.1: Validação com 32 eléctrodos - 1000Hz - 10 1000 2000 ms

7.1.2 Validação com 62 eléctrodos - relógio actualizado a 1000Hz

Nesta validação incrementamos o valor de eléctrodos para 62, mantendo todos os outros parâmetros iguais aos anteriores e podemos observar uma vez mais que todos os eventos se encontram em sintonia com o esperado (figura 7.2).

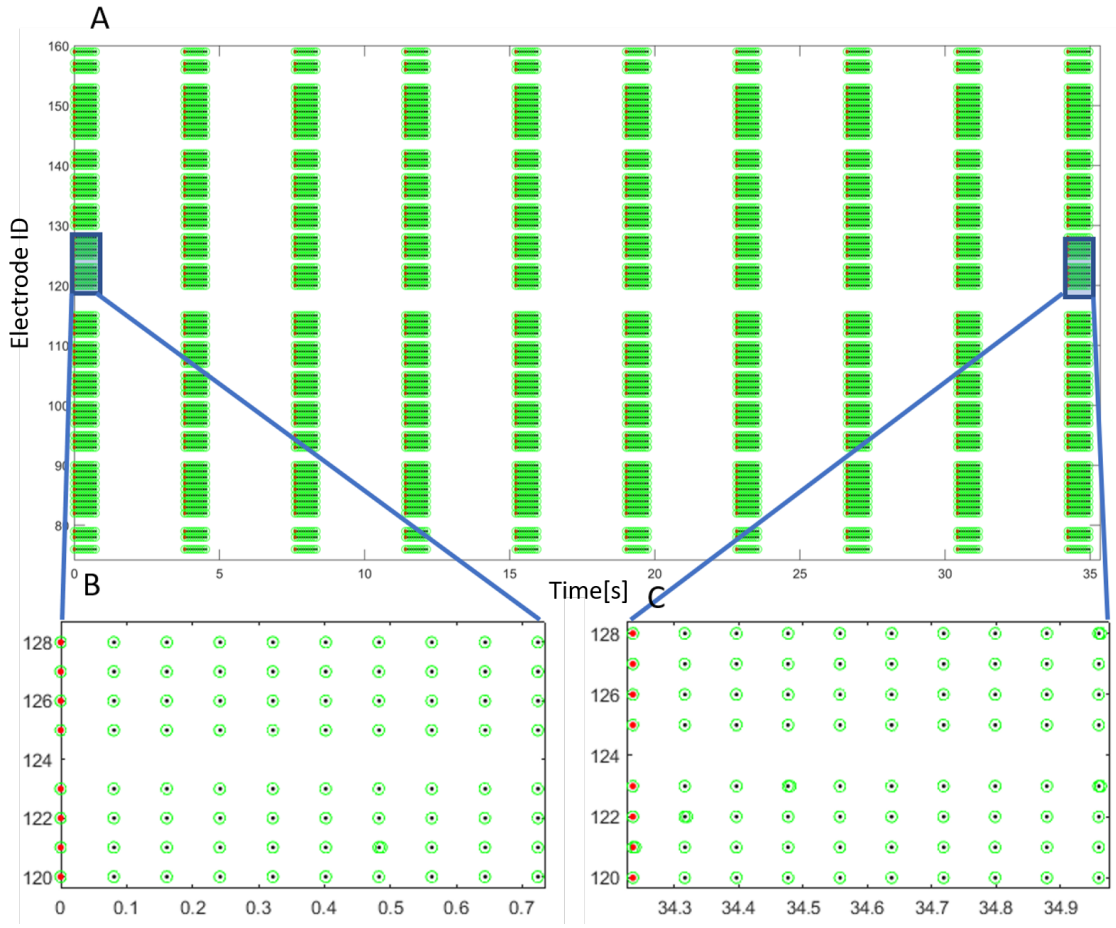

Figura 7.2: Validação para 62 eléctrodos - 1000Hz - 10 1000 2000 ms

7.1.2.1 Validação com 126 eléctrodos - relógio actualizado a 1000Hz

Na validação que se segue os eléctrodos a serem monitorizados passam a ser 126. Dividímo-la em duas partes, uma em que são usados os valores de período de pedido de dados, *spikes*, *bursts* e *network bursts*, por parte do código *C#* de 10, 1000 e 2000 ms, e a outra com valores de 50, 2000 e 5000 ms.

Ao aumentarmos o número de eléctrodos para 126 surgem problemas na actualização do relógio. Todos os eventos aparentam ter sido detectados correctamente, no entanto os instantes em que ocorrem estão, segundo o *DSP* errados, apresentando-se adiantados, como vemos na figura 7.3. Assim, podemos inferir que o relógio não está a ser actualizados à frequência que pretendíamos, mas sim a uma frequência menor, isto deve-se ao facto de começarem a existir conflitos entre os *interrupts*.

Para colmatar este problema decidimos enviar dados para o computador a uma cadência menor, que tem que assegurar que não há perda de eventos por parte do computador. Decidimos colocar os períodos de aquisição de dados a 50, 2000 e 5000 ms no código *C#*.

Ao realizarmos o teste com estas alterações verificamos que o erro diminuiu consideravelmente, no entanto não foi o suficiente para eliminá-lo. Por isso, decidimos avaliar o que acontecia se diminuíssemos a taxa de disparo dos *spikes*, como podemos ver na figura 7.4. Como seria de esperar, desta forma os valores voltam a estar alinhados com os do *Experimenter*. O que quer dizer que não só o número de eléctrodos tem influência na performance do sistema, mas também a actividade de cada eléctrodo.

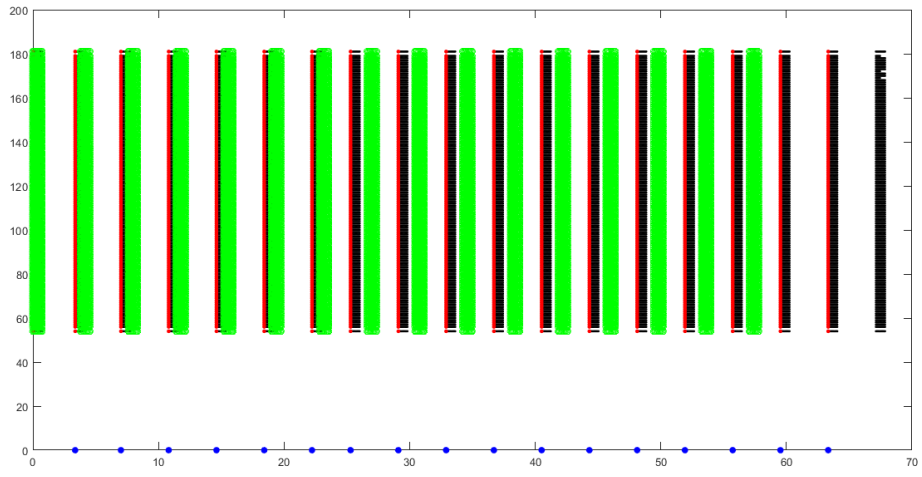

Figura 7.3: Validação para 126 eléctrodos - 1000 Hz - 10 1000 2000 ms

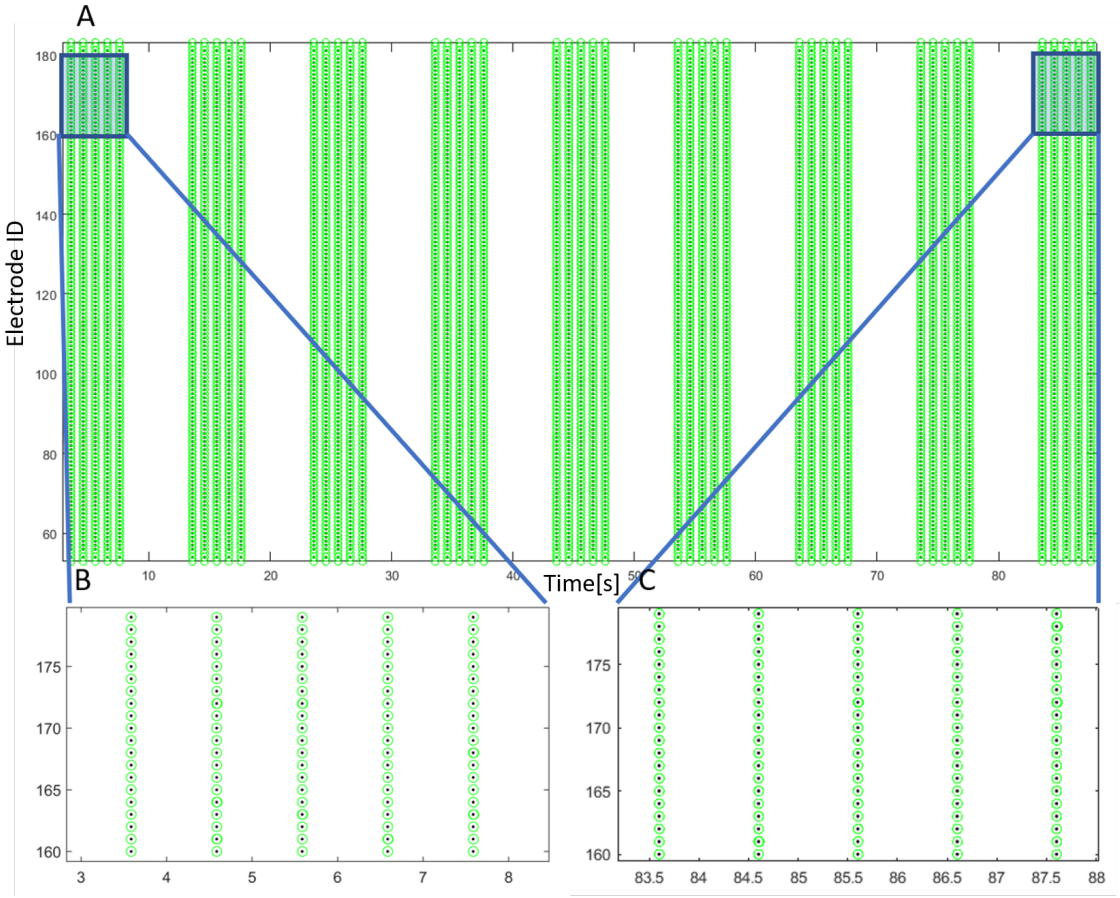

Figura 7.4: Validação para 126 eléctrodos - 1000 Hz - 50 2000 5000 ms

7.1.3 Validação com 62 eléctrodos - relógio actualizado a 2000Hz

Neste caso, aumentámos a frequência do nosso relógio para o dobro de modo a que ele tenha uma maior precisão, assim tem um incremento de 0.0005 segundos de cada vez que é chamado o seu *interrupt*. Na figura 7.5 podemos ver que mais uma vez obtivemos um bom resultado.

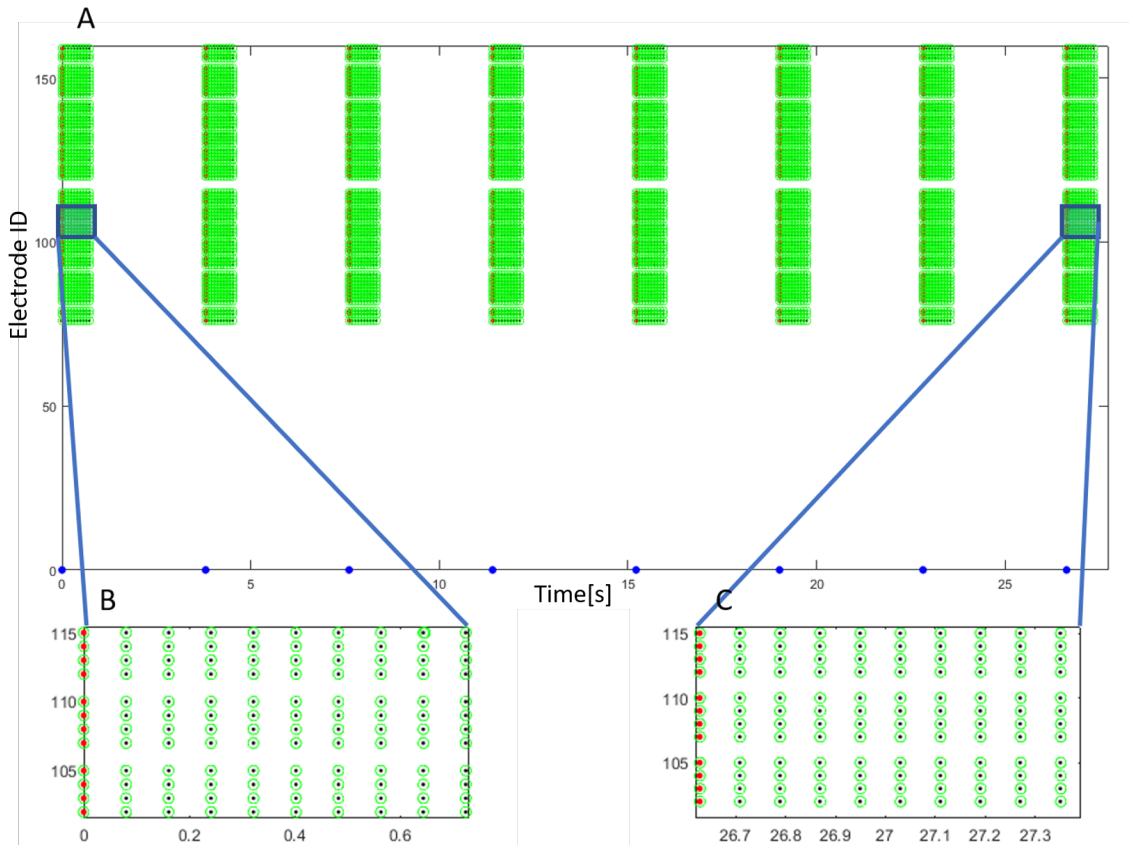

Figura 7.5: Validação com 62 eléctrodos - 2000Hz - 10 1000 2000 ms

7.1.4 Validação com 126 eléctrodos - relógio actualizado a 2000Hz

Esta validação tem a mesma estrutura que a anterior, para 126 eléctrodos e 1000 Hz de actualização de relógio, na qual também alterámos os períodos de aquisição de eventos por parte do computador.

Como seria de esperar temos também um erro associado ao relógio do *DSP*, que ainda se intensifica porque agora estamos a utilizar um período menor de actualização do relógio.

Utilizando a estratégia de aumentar o período dos pedidos por parte do *PC* os nossos resultados melhoram bastante neste caso, no entanto podemos ver na figura 7.6 que existe ainda um *shift* de tempo em relação ao dados retirados do *Experimenter* e dos dados do *DSP*. Pelo que não deve ser usada esta frequência de relógio para este número de eléctrodos.

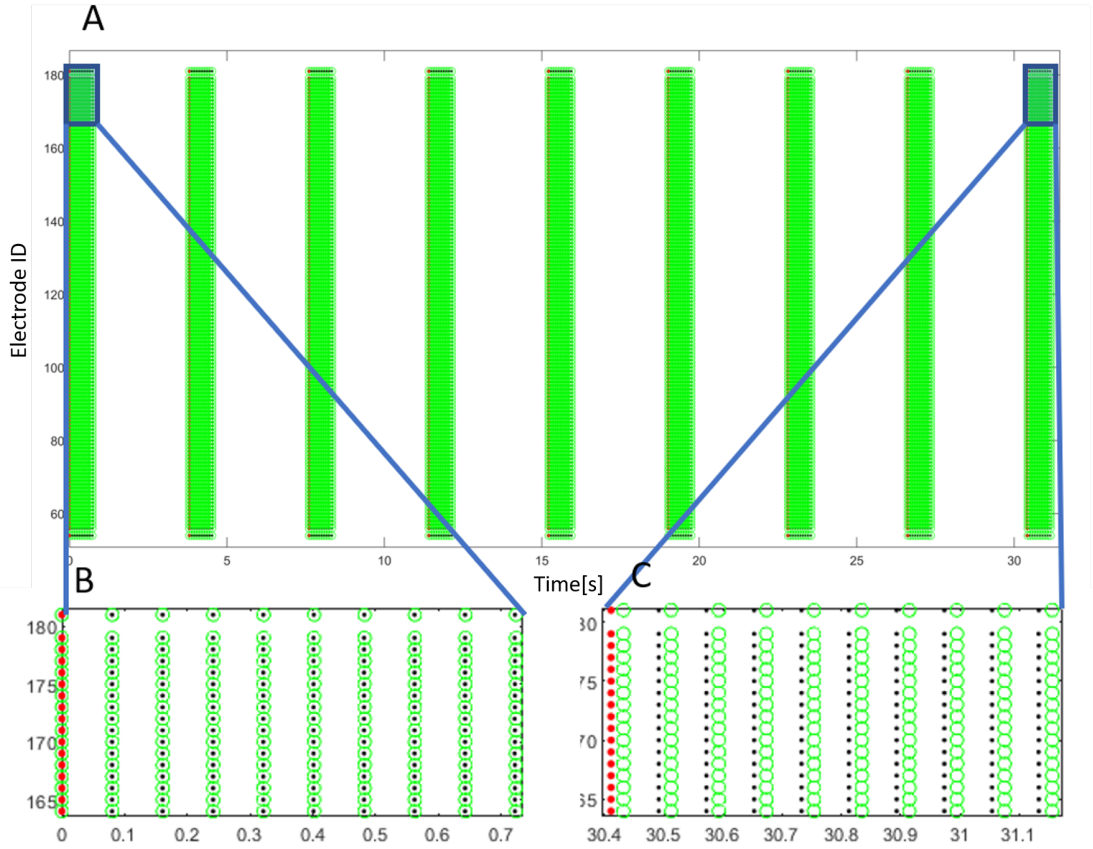

Figura 7.6: Validação para 126 eléctrodos - 2000 Hz - 50 2000 5000 ms

7.2 Análise de performance do sistema

O objectivo desta secção é quantificar o erro associado ao instante a que cada *spike* corresponde consoante a frequência a que estes ocorrem e o número de eléctrodos monitorizados. Poderão existir erros de precisão, ou então atrasos no nosso relógio devido a um aumento do número de eléctrodos a serem monitorizados a detectarem um elevado número de eventos.

Sendo o nosso relógio actualizado à frequência de 1000 Hz, sabemos que o instante a que cada *spike* vai estar associado irá ter uma precisão de 1 ms. Já a aquisição de dados por parte do *DSP* é feita a 50 kHz, logo os instantes das nossas detecções irão ser gravadas com um erro, no máximo de 1 ms em relação ao instante em que realmente ocorreu, que poderá ser positivo ou negativo, ou seja, um *spike* que seja detectado, vai parecer atrasado ou adiantado devido a este erro de precisão.

À medida que começamos a monitorizar mais eléctrodos, os nossos algoritmos de detecção irão demorar mais tempo a correr, o que se irá traduzir num atraso da actualização do nosso relógio, fazendo com que os instantes de tempo associados a eventos pareçam estar adiantados ao que seria de esperar.

Para fazermos a verificação destes erros foi feita a estimulação de canais a 1, 2, 5, 10 e 20 Hz, criando 100 *spikes* para cada taxa de disparo, sendo que fizemos esta experiência incrementando o número de linhas de eléctrodos do *MEA*, uma a uma, a serem monitorizadas. Começámos com uma linha (16 eléctrodos) até um total de 8 linhas (128 eléctrodos). Para analisarmos os resultados fizemos um *script* em *MATLAB* que nos diz qual é o erro associado a cada detecção tendo em conta o número de eléctrodos a monitorizar e a sua taxa de disparo. Verificamos qual era o desvio das diferenças entre *spikes* consecutivos e depois integramos estes desvios para termos acesso ao erro acumulado associado a cada *spike*, para uma dada taxa de disparo e número de canais a serem monitorizados. Fizemos também a análise do valor médio do erro para cada taxa de disparo.

7.2.1 Erro associado à monitorização de 48 eléctrodos

Na figura 7.7 pode-se ver o que foi feito na primeira fase desta verificação, para 48 eléctrodos[1]. A azul temos os desvios associados ao *DSP* e a laranja ao *Experimenter*, que serão sempre nulos, como seria de esperar, já que as detecções não são feitas em tempo real. Cada patamar corresponde ao período a que os *spikes* estão a ser gerados, ou seja, o erro irá oscilar à volta de cada período, sendo o ideal que estes erros se anulem para que não exista propagação de erro. Neste caso podemos observar que não existe qualquer propagação de erro, já que as oscilações observadas se anulam entre si. O eixo das ordenadas representa a contagem de *spikes* que foram gerados, assim de 100 em 100 *spikes* podemos ver que o comportamento do erro se altera, pois alteramos as frequências a que estes estão a ser gerados.

Na figura 7.8 conseguimos ver qual é a acumulação de erro associada a cada *spike*. Analisando a imagem pudemos verificar que estas oscilações se devem apenas a desvios associados à precisão do nosso relógio e não a atrasos do relógio. Já que no final destes testes o erro acumulado é zero.

[1] Esta verificação foi feita para um número de eléctrodos inferior, no entanto apenas iremos mostrar os passos desta a partir deste número de eléctrodos, já que os resultados anteriores são idênticos

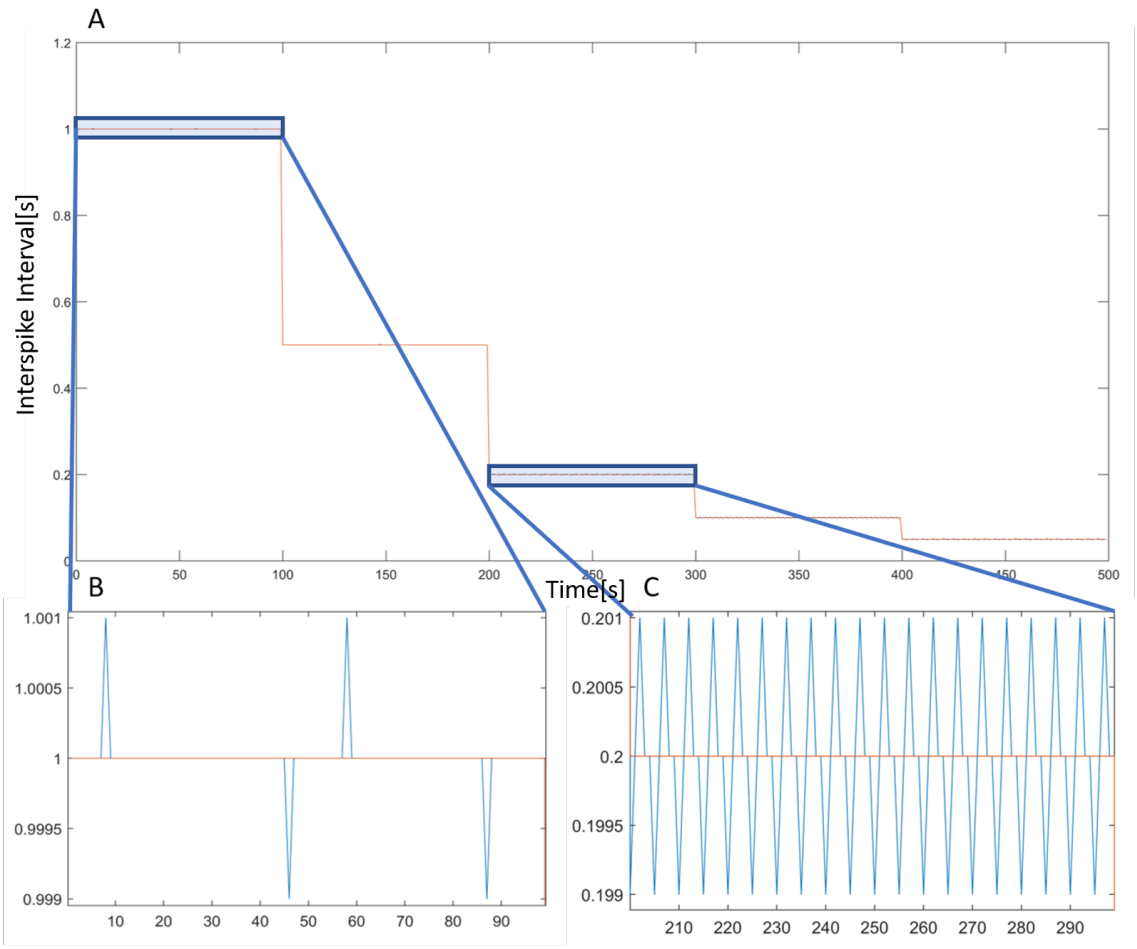

Figura 7.7: Desvios associados a cada frequência com 48 eléctrodos

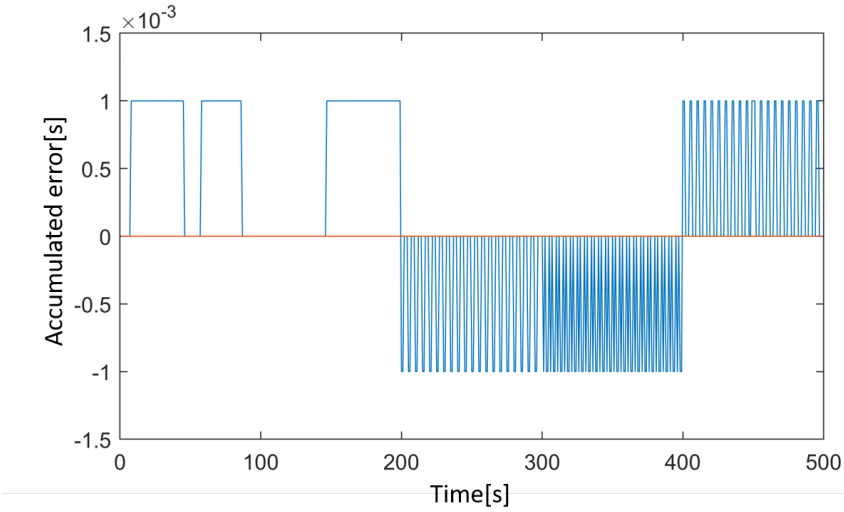

Figura 7.8: Erros acumulado com 48 eléctrodos

7.2.2 Erro associado à monitorização de 64 a 96 eléctrodos

Fazemos agora o mesmo processo para a verificação do erro com 64 eléctrodos. Na figura 7.9 já conseguimos ver que, para este número de eléctrodos a serem detectados em simultâneo, já temos alguns problemas relativos a atrasos do relógio. O que significa que se tivermos 64 eléctrodos em simultâneo a fazerem detecções de *spikes* que ocorrem a uma taxa de 20 Hz iremos ter problemas na gravação dos instantes de tempo das detecções de eventos (mas não na detecção em tempo real dos *spikes* propriamente ditos).

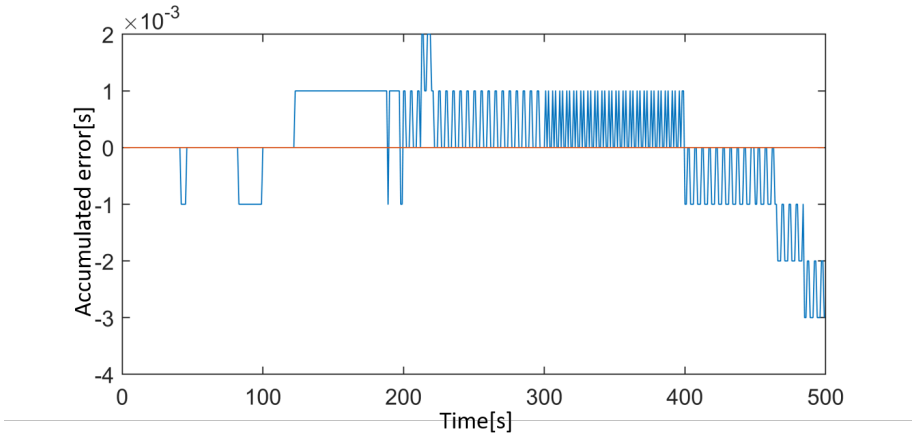

Figura 7.9: Erros acumulado com 64 eléctrodos

Prosseguimos então nestas verificações para 80 eléctrodos, sendo que aqui surgiu um novo problema, que nos fez aperceber que existe um erro no nosso código de comunicação. Quando atingimos os 20 Hz seria de esperar que o valor final de erro acumulado fosse maior que na verificação anterior, no entanto apercebemo-nos que existia um salto brusco do erro nesta frequência. A explicação que encontramos para este fenómeno está associada a um erro de comunicação que ocorre quando os registos da *mailbox* não são suficientes para que o envio dos dados seja feito de uma só vez, verificando o instante em que ocorria esta situação e comparando com o instante dos *spikes* que causam esta variação brusca no erro. Na figura 7.10 temos representada esta situação à frequência de 20 Hz.

Apesar de este erro piorar os resultados da nossa análise, podemos analisar todos os dados recolhidos antes que ele ocorra. Na figura 7.11 comparamos o erro acumulado entre medições feitas monitorizando 64 e 80 eléctrodos. A diferença é notória entre ambas as situações, já que o erro, para 80 eléctrodos, se começa a propagar mais cedo, sendo que chega a um mesmo valor de acumulação do mesmo em menor espaço de tempo e além disso apresenta, também, o problema de os dados das detecções excederem a memória da *mailbox*.

Na figura 7.12 podemos ver novamente que a tendência da acumulação de erro se deteriora para mais eléctrodos e em frequências cada vez mais baixas. Com 96 eléctrodos a serem monitorizados já temos um erro bastante significativo a 10 Hz, sendo que o erro obtido em relação à

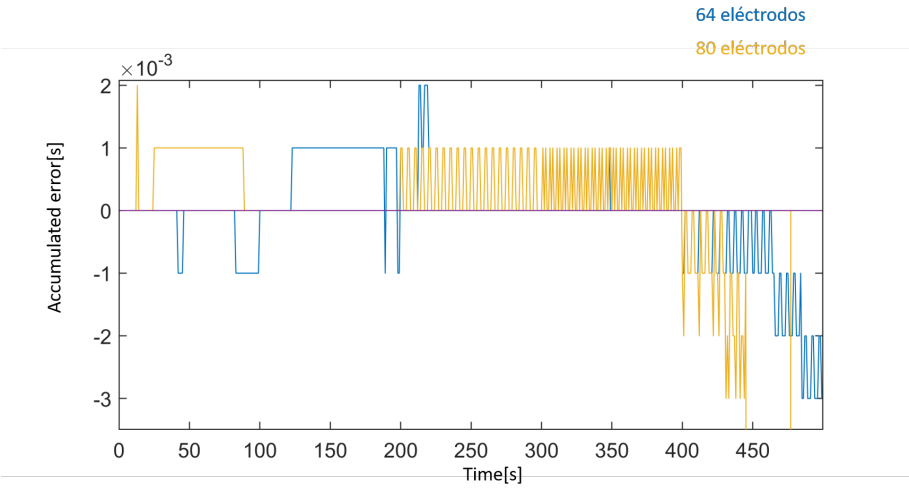

Figura 7.10: Erro acumulado com 80 eléctrodos

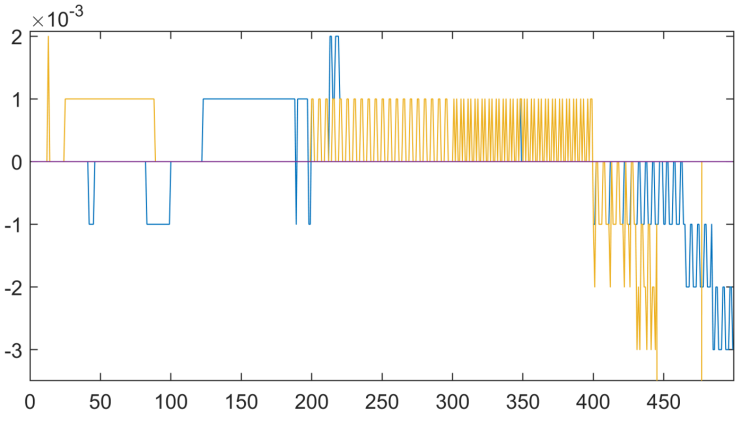

Figura 7.11: Comparação do erro acumulado entre 64 e 80 eléctrodos

verificação anterior também se propaga mais cedo e o problema de envio de dados ocorre também antecipadamente.

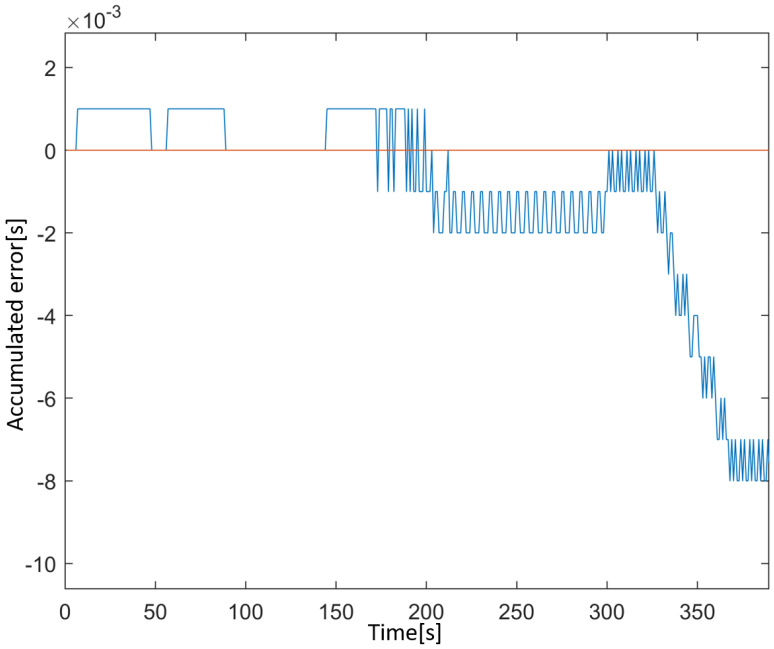

Figura 7.12: Erro acumulado com 96 eléctrodos

7.2.3 Média do erro com o aumento de número de eléctrodos e frequência

Para finalizarmos estes testes, de modo a termos informação mais concreta e facilmente comparável, decidimos calcular, para cada número de eléctrodos a detectarem *spikes*, de 48 a 96 (incrementando 16 eléctrodos de cada vez), qual era a média do erro para cada taxa de disparo de *spikes* (1, 2, 5, 10, 20 Hz). Isto foi feito através de um gráfico que representa três dimensões, nas abcissas está representado o número de eléctrodos a serem testados, nas ordenadas as frequências a que foram gerados os *spikes* e a terceira dimensão, o erro médio para cada taxa de disparo e número de eléctrodos, é representado por uma escala de cores.

Feito isto, na figura 7.13, podemos ver qual a influência que estes factores têm no erro da gravação dos tempos das detecções. Observando esta figura chegamos à conclusão que apenas existem erros consideráveis quando existe um número elevado de eléctrodos com actividade a elevadas frequências, o que é pouco comum, pelo que esse problema dificilmente surgirá. Ainda assim, caso exista uma elevada actividade neuronal que ponha em causa as gravações conseguimos ter noção de qual será o impacto dessa actividade nos instantes gravados.

Podemos também notar que para 48 e 64 eléctrodos nas frequências 5 e 10 temos um certo erro associado, inferior a 1 ms, que atribuímos ao erro associado à precisão do relógio e não a um atraso real, já que na frequência de 20 Hz este "erro"volta a ser nulo. Para confirmarmos esta afirmação

bastou-nos comparar os dados gravados através da monitorização do *DSP* e do *Experimenter*, que nos mostram exactamente que não há qualquer atraso do relógio, mas sim erros de precisão.

O quadrado a preto que podemos ver na imagem de-se apenas ao facto do valor que estava ali representado não fazer sentido devido ao problema que temos na comunicação com a *mailbox* no caso do registos não terem espaço suficiente para o envio da informação de uma só vez.

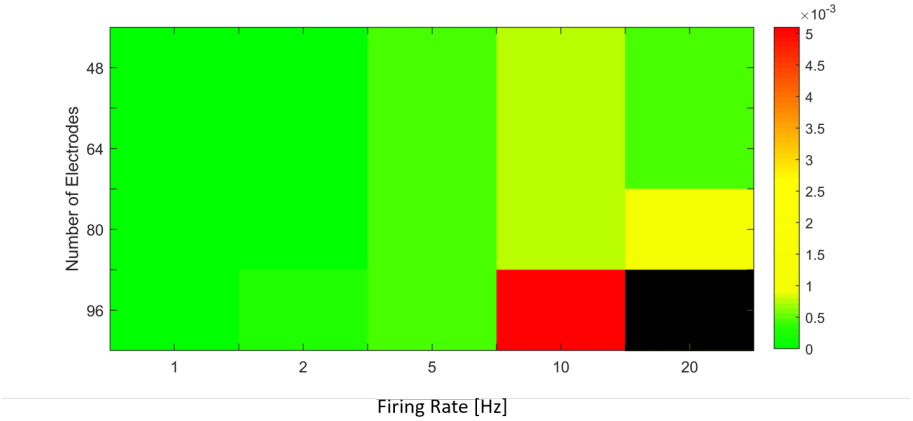

Figura 7.13: Média do erro com o aumento do número de eléctrodos e frequência

7.3 Filtragem dos Sinais

Ao lermos a actividade eléctrica das células através do *MEA* podemos verificar que existe uma grande componente de baixa frequência dos sinais, sendo assim, é essencial que seja feita um filtragem prévia destas frequências para que as leituras não fiquem comprometidas.

Esta filtragem é configurada através da escrita dos coeficientes do filtro em registos específicos da *interface board* do *MEA2100*. Para o cálculo destes coeficientes é utilizado um método presente numa classe que se encontra na *DLL* disponibilizada pela *MCS*, na qual podemos escolher o tipo de filtro a ser utilizado, qual a banda passante, a frequência de corte e também a ordem do filtro. Os coeficientes obtidos são posteriormente enviados para os registos de configuração do *MEA* correspondentes ao filtro antes do *DSP* começar a adquirir dados.

O tipo de filtro que escolhemos utilizar é o *Butterworth*, pois este é um filtro que apresenta, na banda passante, a resposta em frequência o mais plana possível, ou seja não altera o ganho do sinal nas frequências que o caracterizam. Apenas utilizamos um filtro passa alto, para filtrar as componentes de baixa frequência, de ordem dois, que se mostrou bastante eficaz para o nosso fim.

Para verificarmos se o filtro que se configurou estava de facto a funcionar, no *Experimenter* programámos três estímulos e comparámos os sinais obtidos com e sem o filtro. Na figura 7.14 a azul está representado o sinal sem filtro, a preto e a vermelho os sinais filtrados com uma frequência de corte de 100 Hz, sendo que o preto foi filtrado configurando o filtro a partir do *Experimenter* e o outro através do código do nosso sistema. Como é fácil de ver, os sinais são bastante semelhantes, o que quer dizer que a configuração feita pelo nosso sistema está correcta e podemos

começar a adquirir os sinais dos neurónios. No entanto, podemos também ver, na imagem C, que o sinal do *Experimenter* tem uma amplitude ligeiramente para a mesma frequência de corte, o que pode justificar algumas diferenças entre o *DSP* e o *Experimenter* quando a amplitude dos sinais estiverem próximas do valor de *threshold*.

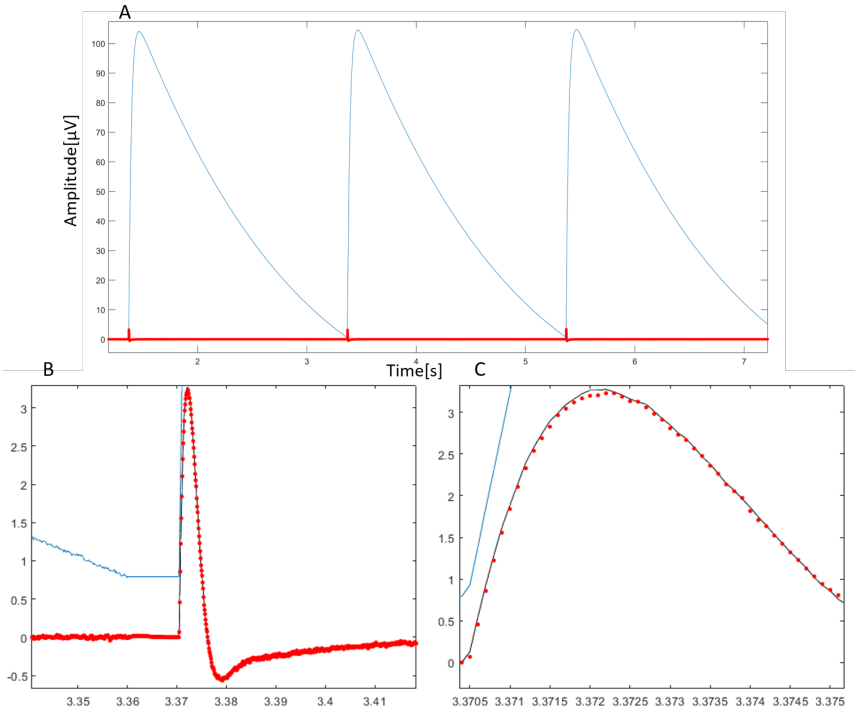

Figura 7.14: Filtragem de estímulos

7.4 Monitorização de actividade neuronal

Depois de validados com sinais sintéticos gerados pelo estimulador eléctrico (secção 7.1), os algoritmos de monitorização e de comunicação foram testados com culturas de neurónios. Usaram-se dois tipos diferentes de culturas neuronais - populações de neurónios hipocampais (subsecção 7.4.1) e explantes de ganglios da raiz dorsal (subsecção 7.4.2)- testando assim a eficácia dos algoritmos para monitorização de populações com dinâmicas muito distintas. Todas as experiências com células foram realizadas de acordo com a legislação Europeia para o uso de animais para fins científicos e pelos protocolos aprovados pelo comité ético do i3S.

7.4.1 Monitorização de células (hipocampais) utilizando *threshold* manual com o relógio a 2000Hz

As células hipocampais foram extraídas de embriões de ratos (E18) e depositadas sobre a câmara do *MEA*, evolvendo a matriz de micro-eléctrodos. Após vários dias *in vitro*, estas populações de neurónios começam a exibir actividade coordenada, emergindo assim *network Bursts*,

detectados em vários eléctrodos.

De modo a validar os resultados obtidos pelo nosso algoritmo, a detecção de *spikes* foi também analisada no *Experimenter* (não em tempo real) com os mesmos parâmetros utilizados no código do *DSP* - *threshold* de 200uV e *deadtime* de 3 ms. No entanto a taxa de aquisição no caso do *Experimenter* foi de 10kHz, sendo diferente do utilizado pelo *DSP*, 50kHz, o que pode justificar eventuais diferenças na detecção de *spikes*. Escolhemos em ambos os softwares monitorizar 96 eléctrodos, na figura 7.15 temos a sobreposição entre as detecções de *spikes* do *Experimenter* (a verde) e do *DSP* (a preto) e ainda as detecções de *bursts* de eléctrodos individuais (a vermelho) e *network bursts* (a azul), calculados pelo DSP em tempo real. Existem alguns casos em que o *DSP* detecta eventos e o *Experimenter* não e vice-versa. Podemos atribuir estes erros às diferentes taxas de aquisição, a filtragem dos sinais ter um resultado diferente nos dois softwares e a *spikes* que estão muito próximos do valor do *threshold*, que em certos casos são detectados num software e não no outro devido aos factores explicitados anteriormente.

Nesta primeira verificação decidimos colocar a taxa de actualização do relógio a 2000 Hz de modo a percebermos se conseguimos fazer monitorização de actividade neuronal com o relógio superior a 1000Hz. Os resultados obtidos mostram-nos que é perfeitamente possível fazê-lo, vamos ter uma ligeira melhoria na precisão do relógio, no entanto com um maior risco do relógio se atrasar mais facilmente. Por esse motivo, decidimos manter o relógio nos 1000Hz nas próximas verificações para que esse risco seja menor.

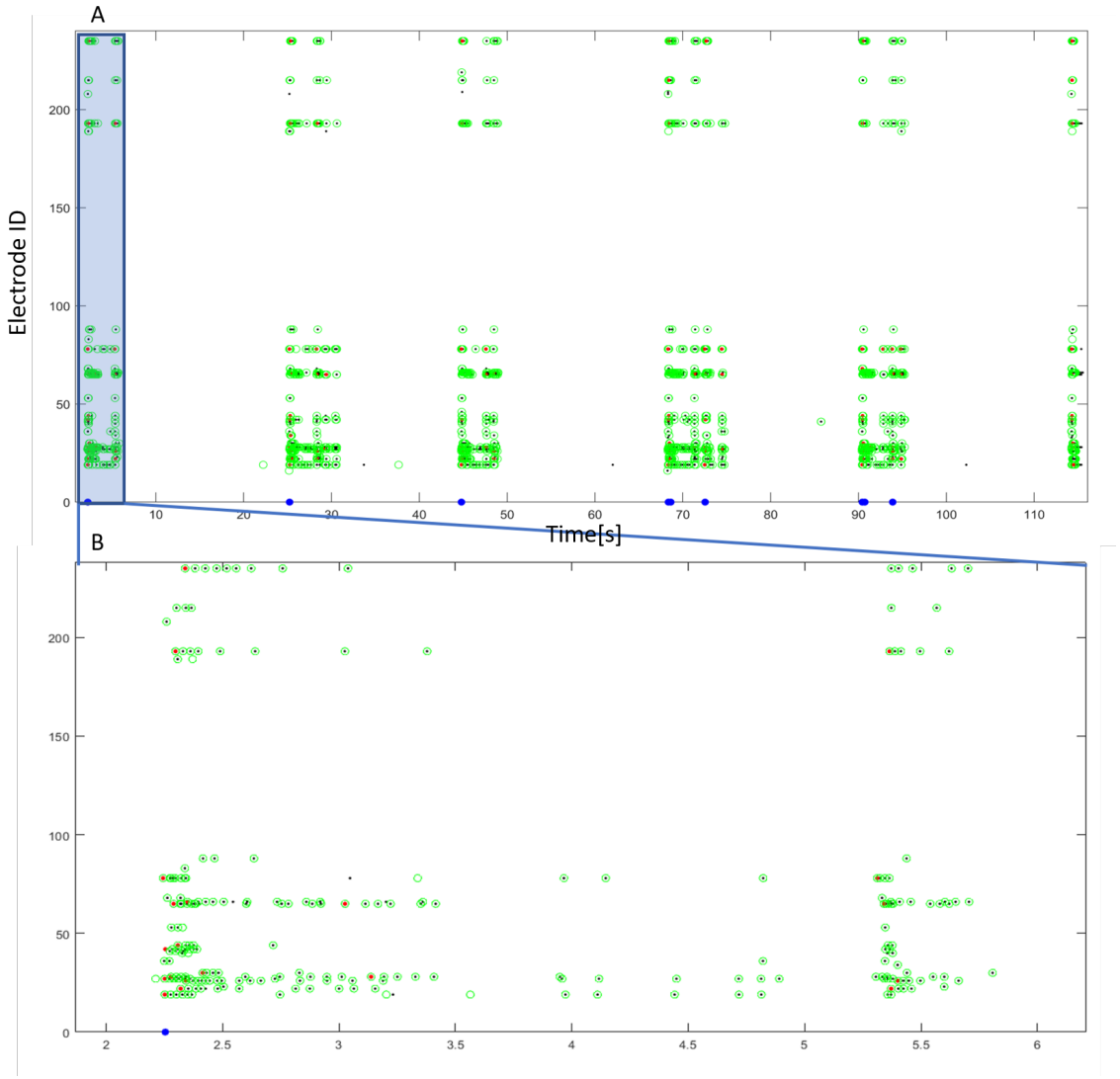

Figura 7.15: Monitorização de células neuronais (hipocampais) com 96 eléctrodos - com actualização de relógio a 2000Hz - com *threshold* manual

7.4.2 Monitorização de células (*DRGs*) utilizando *threshold* manual com o relógio a 1000Hz

Para esta experiência usámos gânglios da raiz dorsal (*dorsal root ganglion, DRG*) de embriões de ratos. Estes nódulos contêm aglomerados de neurónios responsáveis pela propagação de informação sensorial até ao sistema nervoso central.

Os explantes de *DRGs* foram colocados numa câmara micro-fluídica, fixada no topo do *MEA*. Estas câmaras contém 2 compartimentos: compartimento somal (figura 7.16, topo à esquerda) e axonal (figura 7.16, base à esquerda). Entre os dois compartimentos encontram-se micro-canais, cada um alinhado com uma coluna de micro-eléctrodos. Os *DRGs* foram colocados no compartimento somal da câmara micro-fluídica. Após vários dias *in vitro* os seus axónios expandem-se,

atravessam os micro-canais, chegando assim ao compartimento axonal. Devido às reduzidas dimensões dos micro-canais, estes contém apenas axónios, provenientes de diferentes neurónios dos *DRGs*.

Assim, os micro-eléctrodos posicionados debaixo dos micro-canais captam a propagação dos potenciais dos acção dos axónios. É fácil perceber que sempre que existir actividade eléctrica nas células pertencentes a uma determinada coluna, então deveremos detectar actividade em todos os eléctrodos da mesma. Neste caso decidimos monitorizar 126 eléctrodos de modo a percebermos se é possível fazermos detecção em tempo real para esta quantidade de eléctrodos sem danificar as gravações dos instantes de tempo das detecções.

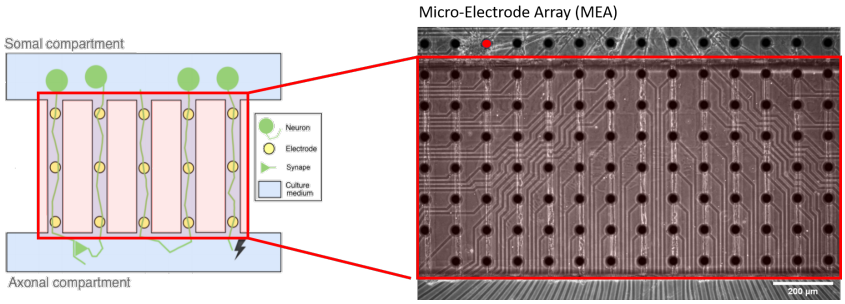

Figura 7.16: *MEA* utilizado com câmaras micro-fluídicas (adaptado de [53])

Na figura 7.17A verificamos que determinadas colunas do *MEA* de facto se comportam como explicámos anteriormente, sendo que apenas três se encontram em actividade intensa. Já na figura 7.17B podemos ver com mais detalhe a forma do sinal de um dos eléctrodos dessas colunas e verificamos que os potenciais de acção estão ocorrer a uma cadência praticamente constante, apenas com certas flutuações de um período médio. Com esta monitorização pudemos também provar que é possível monitorizarmos um número considerável de eléctrodos com alguns deles com actividade, relativamente constante, a frequências entre os 20 e os 30 Hz sem que tenhamos um atraso do relógio ou comportamentos indesejados.

Assim, mesmo antes de realmente verificarmos os *spikes* de cada coluna de eléctrodos podemos inferir que, fazendo um *interspike histogram*, nas colunas onde existe este tipo de actividade o histograma terá uma forma gaussiana, devido a alta consistência da taxa de disparo ao longo dos diferentes eléctrodos do canal.

Foi feita, então, uma detecção de actividade neuronal com o *DSP*, sendo que na interface de utilizador escolhemos visualizar os *interspike histograms* dos dados que iam sendo recebidos, monitorizando 126 eléctrodos. Neste caso, a cada coluna do histograma atribuímos um intervalo de tempo de 0.001 segundos. Os resultados obtidos, presentes na figura 7.18, apresentam-se conforme o esperado.

Enquanto os 126 eléctrodos eram monitorizados pelo *DSP*, também o eram por um outro computador através do *Experimenter*, que fez a sua gravação, de modo a podermos comparar os dados recolhidos através de um *script* em *MATLAB*. Para que os dados comparados fizessem sentido tivemos que definir os *thresholds* manualmente em ambos os *softwares*, de maneira a que

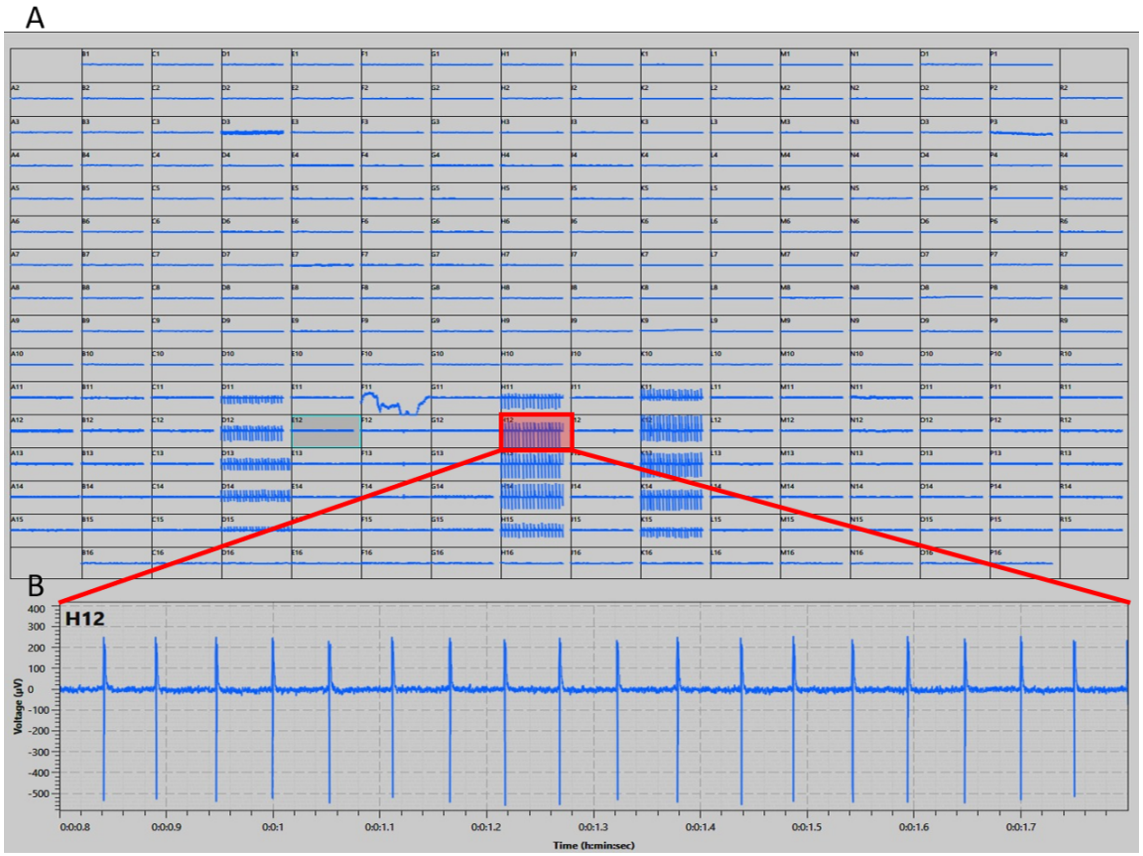

Figura 7.17: Visualização dos sinais no *Experimenter*

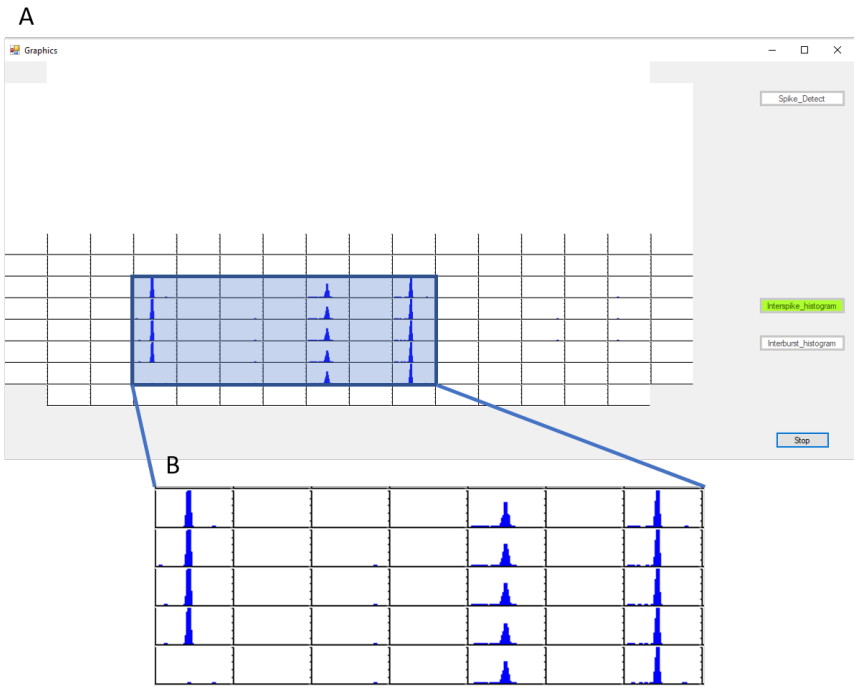

Figura 7.18: *Interspike histogram* da *User Interface*

detectássemos exactamente os mesmo *spikes* das duas formas. Na figura 7.19 vê-se através dos círculos a vermelho os dados exportados do *Experimenter* e os pontos a preto representam os dados que foram recebidos pelo *DSP*. Após a análise destes *raster plots* podemos verificar que os resultados para esta medição foram bastante satisfatórios, no entanto existem alguns eléctrodos em específico que detectaram eventos no *Experimenter* e não no *DSP*, por isso tivemos que fazer uma análise mais cuidadosa destes casos.

Para se verificar o que acontecia nestas situações decidiu-se analisar os valores de tensão da gravação feita em eléctrodos específicos. Sobrepusemos o sinal em tensão, de um eléctrodo em que falhavam alguns *spikes* ao *DSP*, com o valor de *thresold* definido, 40 uV, e mais uma vez, nos círculos a vermelho vemos os dados do *Experimenter* e a preto os do *DSP*. Pode-se, então, observar na figura 7.20 que os *spikes* mais à esquerda na verdade representam ruído de fundo de alta frequência, e que os *thresholds* definidos se encontram no limiar da detecção deste ruído. Ou seja, no caso dos *thresholds* estarem demasiado próximos do valor de tensão do ruído podem gerar-se este tipo de problemas, pelo que o melhor será sempre utilizar o *auto threshold* para definir estes valores.

No caso de canais em que o *threshold* está suficientemente afastado do ruído este problema deixa de existir e apenas são detectados *spikes* reais de ambos os lados. Na figura 7.21 temos um bom exemplo deste caso.

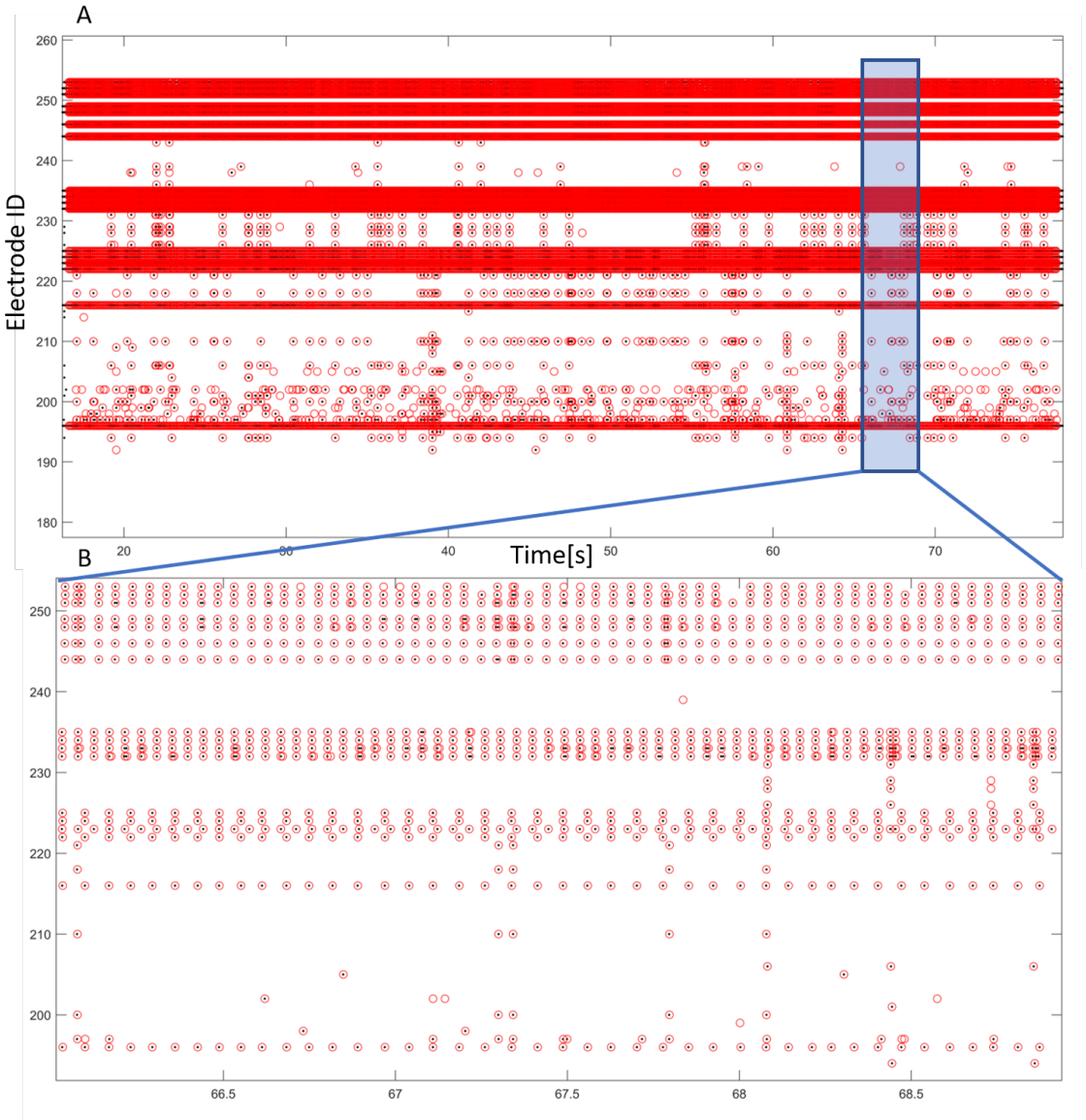

Figura 7.19: Monitorização de células neuronais (*DRGs*) com 126 eléctrodos - com actualização de relógio a 1000Hz - com *threshold* manual

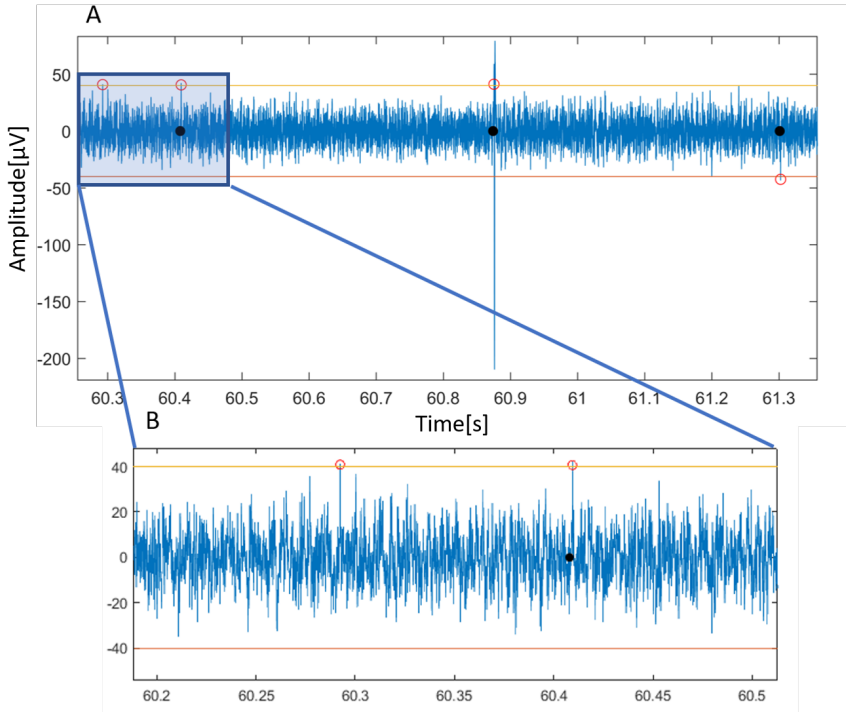

Figura 7.20: *"Spikes"* não detectados

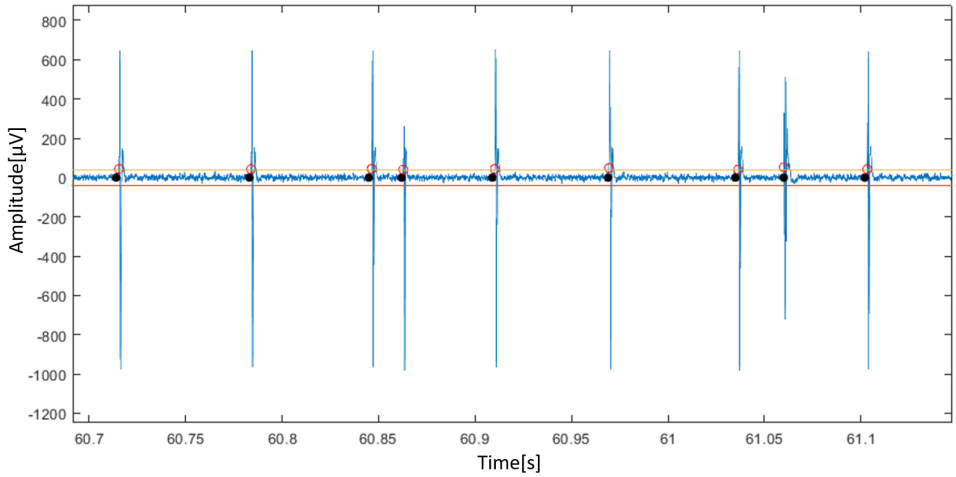

Figura 7.21: *Spikes* detectados

7.4.3 Monitorização de células utilizando *threshold* manual com o relógio a 1000Hz - 6 well

Nesta monitorização voltámos a utilizar células hipocampais extraídas de ratos (E18), com a diferença de que utilizámos um tipo de *MEA* diferente: o *MEA 6-Well*, que podemos ver na figura 7.22. Esta matriz de eléctrodos é dividida em seis zonas independentes (normalmente chamadas de poços - com 42 eléctrodos cada) de modo a que os investigadores possam comparar diferentes comportamentos em cada uma destas zonas. Neste caso decidimos apenas monitorizar um dos poços presentes no *MEA* já que verificamos no *Experimenter* que apenas havia actividade num dos poços, como podemos ver na figura 7.23.

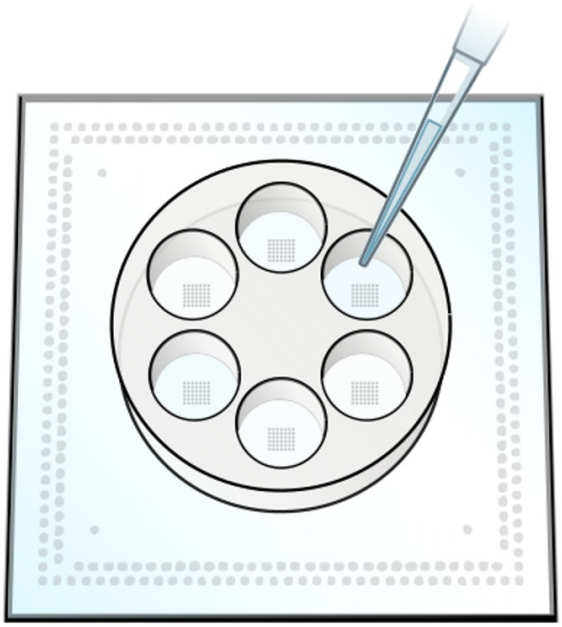

Figura 7.22: *MEA* 6-Well

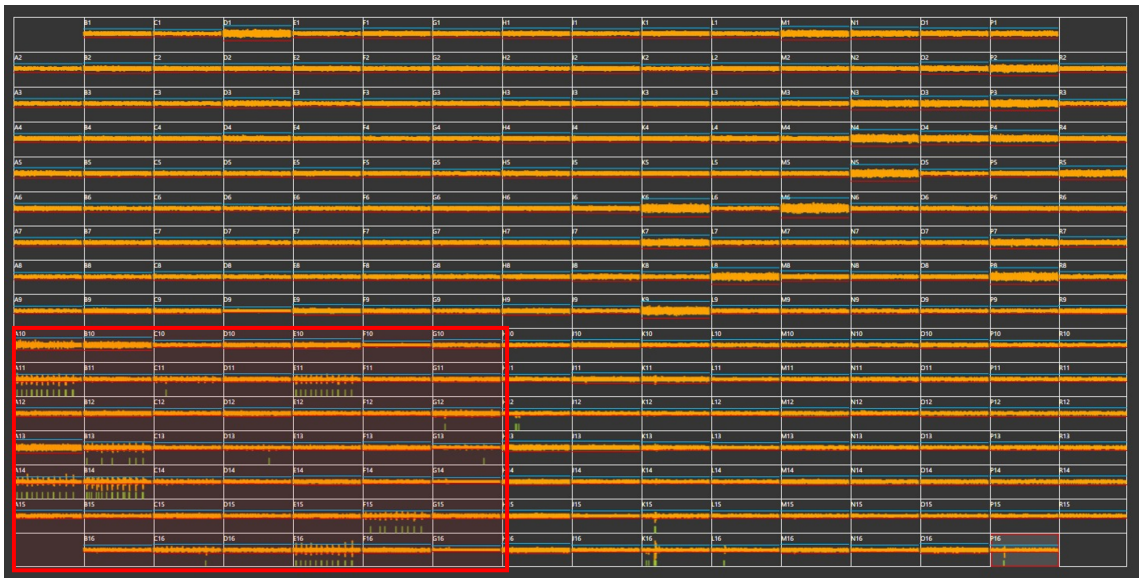

Figura 7.23: Visualização actividade neuronal em todo o *MEA* no *Experimenter*

Mais uma vez, na figura 7.24, podemos verificar que as detecções feitas no *DSP*, em tempo real, são bem sucedidas sem qualquer tipo de atraso no relógio. Podemos também perceber que naqueles eléctrodos em que há apenas algumas detecções esporádicas o *DSP* nem sempre detecta os *spikes* que *Experimenter* detecta, isto deve-se, novamente, ao facto dos *spikes* estarem muito próximos do *threshold* que podem não ser detectado pelo *DSP*.

Na figura 7.25 pode ver-se a os eventos detectados pelo *DSP* que vão sendo enviados para o computador através do *raster plot* da *user interface*. A azul vemos os *spikes* e a vermelho os *bursts*. Nesta janela temporal não foi detectado nenhum *network burst*, que seria representado com um losango roxo na abcissa de valor 0. O eixo das abcissas corresponde ao *ID* do eléctrdo e as ordenas correspondem ao tempo de gravação em segundos. Podemos ver que no início da gravação são detectados 3 *spikes* seguidos em todos os eléctrodos monitorizados. Isto ocorreu porque decidimos enviar três estímulos para cada canal para que na validação fosse mais fácil alinharmos os resultados obtidos no *DSP* e *Experimenter*, já que a gravação dos eventos inicia-se em instantes diferentes em cada um dos *softwares*.

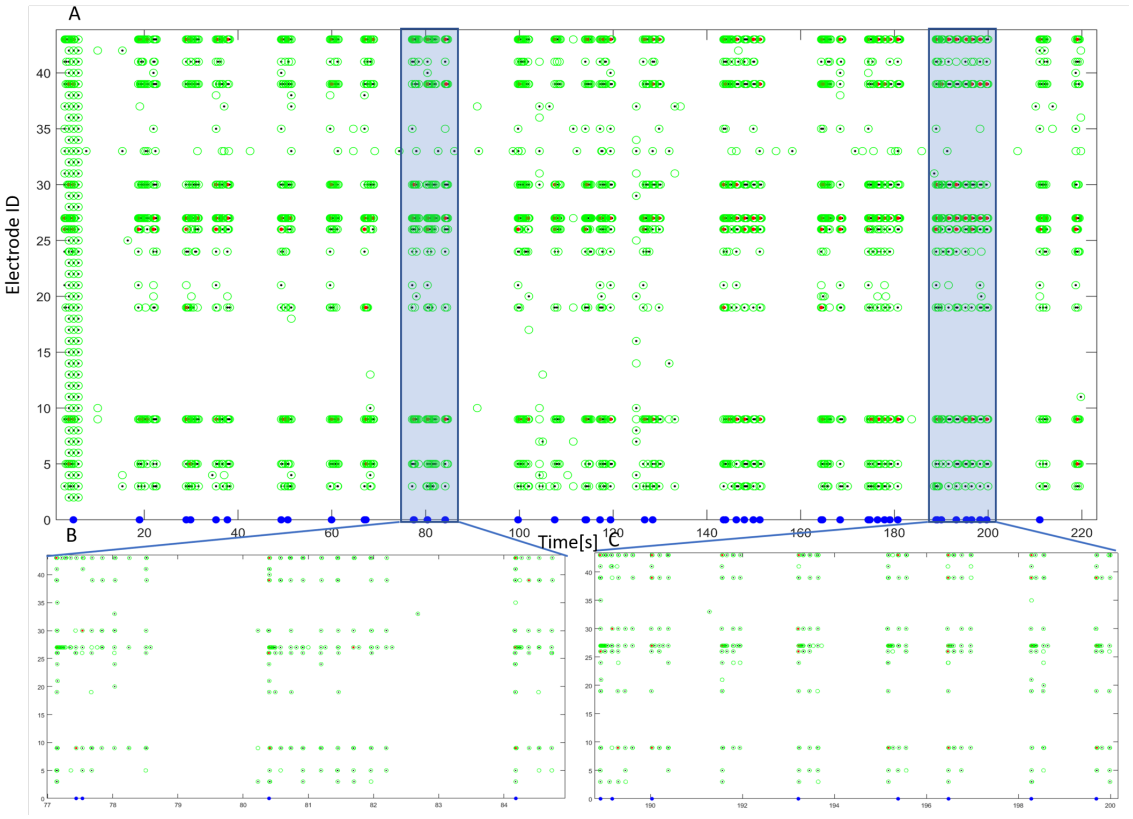

Figura 7.24: Comparação entre dados os recolhidos no *Experimenter* e no *DSP* - com *threshold* manual

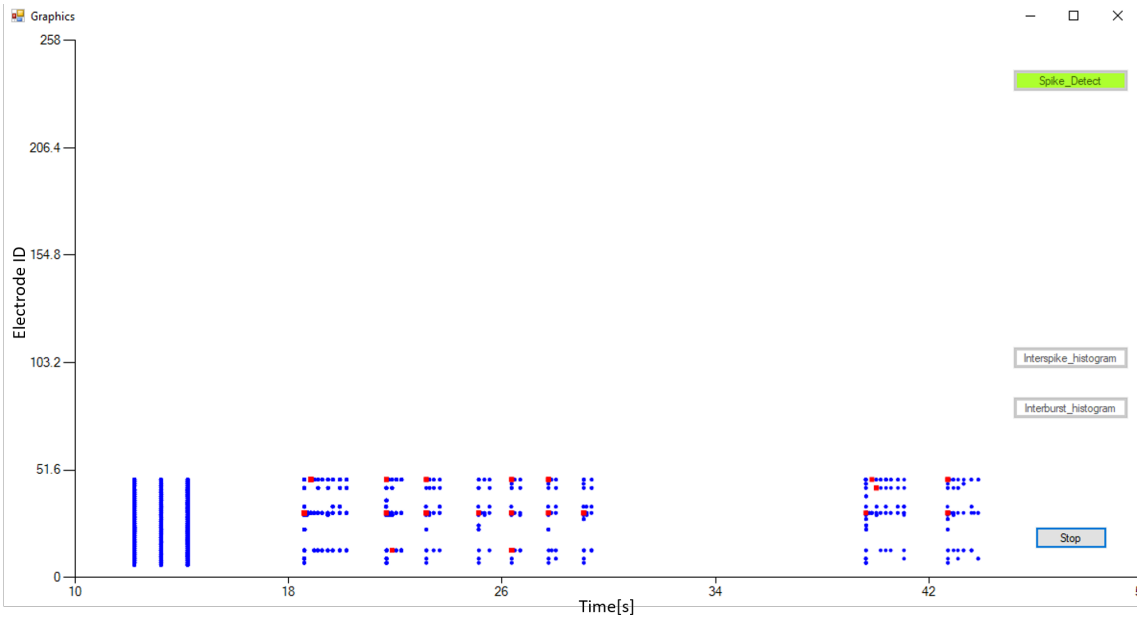

Figura 7.25: Visualização de eventos através de *raster plot* no sistema desenvolvido

7.4.4 Monitorização de células utilizando *threshold* automático com o relógio a 1000Hz - 6 well

Até agora apenas fizemos verificações utilizando *thresholds* definidos manualmente tanto no *DSP* como no *Experimenter* de modo a podermos ter os resultados mais semelhantes possíveis. Partimos então para a análise da detecção dos eventos com o *threshold* definido automaticamente para cada eléctrodo. Os *thresholds* foram calculados definindo o mesmo ganho para ambos os casos (equivalente a seis desvios padrão).

Na figura 7.26 podemos observar os resultados obtidos. Os *trhesholds* do o *Experimenter* e no *DSP* são calculados de forma diferente, sendo que no *Experimenter* é feito utilizando uma amostra de 100 ms e no nosso código este é calculado utilizando uma amostra de 1 s para cada canal, o que justifica a diferença nos resultados obtidos. Olhando para a imagem vemos que há uma diferença bastante acentuada no que toca a alguns canais, com isto em vista sabemos que quando fazemos o *threshold* automaticamente temos que ter este cuidado no lado do *DSP*, já que existem vários eléctrodos que nunca detectam eventos. Neste caso, teríamos que diminuir o ganho dos *thresholds* calculados de modo a resolver esta situação.

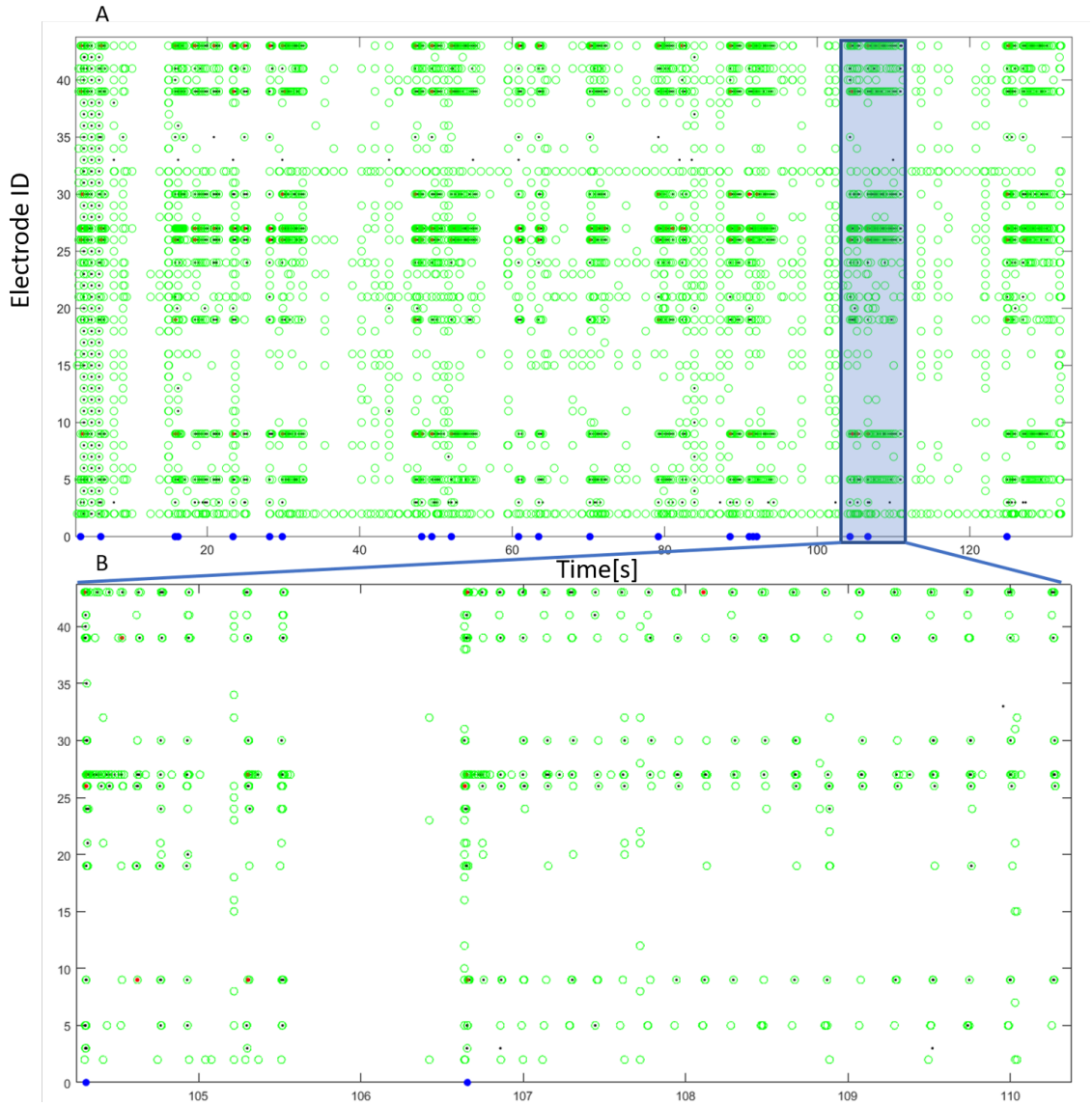

Figura 7.26: Comparação entre dados os recolhidos no *Experimenter* e no *DSP* - com *threshold* automático

7.5 Conclusões

Neste capítulo fomos capazes de saber que influência tem a mudança de frequência do relógio, a frequência de pedidos de dados por parte do computador, a alteração do número de eléctrodos a monitorizar e a frequência dos eventos detectados. Mais importante, conseguimos validar o nosso sistema e perceber que este funciona como era esperado, fazendo detecção de eventos em tempo real enviando essa informação periodicamente para o computador.

Ao aumentarmos a frequência do relógio conseguimos alterar a precisão do tempo atribuído

aos eventos detectados em tempo real com o possível custo de atraso no mesmo em certas situações. Concluímos que o mais indicado é actualizarmos o relógio a uma frequência de 1000Hz. Aumentar o número de eléctrodos que detectam actividade pode, a partir de certo nível de frequência de actividade, introduzir um erro no relógio. Conseguimos também concluir que se diminuirmos a cadência da comunicação entre computador e *DSP* podemos melhorar a performance do sistema.

Por fim, fazendo monitorização de actividade neuronal conseguimos perceber que situações que possam atrasar o nosso relógio são extremamente pouco prováveis, sendo que não detectamos nenhuma situação em que acontecesse (com o relógio a 1000Hz), mesmo com actividade intensa em alguns dos eléctrodos. Sabemos também que podemos monitorizar pelo menos 128 canais sem problemas desde que a rede tenha um nível de actividade "normal"(tal como aconteceu com as diferentes culturas analisadas). É também importante referir que mesmo que o relógio se atrase por algum motivo os eventos são detectados de qualquer das formas em tempo real, pelo que numa aplicação real em malha fechada, sem que esses dados tenham que ser enviados para o computador, o sistema não terá esse tipo de problemas. Aliás, em numa situação onde o foco é pôr o sistema a funcionar em malha fechada, o relógio nem será necessário, uma vez que este serve apenas para verificação dos resultados a posteriori. O mais importante é que seja capaz de fazer as detecções em tempo real.

Capítulo 8

Conclusões

Hoje em dia existe um grande interesse, na área da neuro-ciência e da biotecnologia, de se encontrar novas formas e estratégias de estudar e controlar a actividade neuronal de modo a que possamos tratar doenças, sejam estas motoras, neurológicas ou psicológicas. Existem já formas de extrairmos a informação necessária do cérebro para controlarmos dispositivos externos, no entanto ainda existe um percurso longo no que toca ao controlo em malha fechada destas redes neuronais. Para isso é necessário que consigamos monitorizar a actividade neuronal em tempo real através de sistemas embebidos, que possam ser implantados invasivamente no cérebro, de modo a que seja possível actuarmos sobre estas redes para curarmos doenças como *Parkinson* e epilepsia.

Neste trabalho propusemo-nos a elaborar um sistema capaz de detectar e analisar eventos de actividade neuronal, *in vitro*, em tempo real através de um *DSP* e enviar esta informação para o computador no qual, através de uma *user interface*, é possível visualizar estas detecções à medida que acontecem. Para tal utilizámos um sistema de aquisição de actividade neuronal da *MCS*, *MEA2100-256*, que lê os dados através de um *MEA* com 256 eléctrodos e que contém também um *DSP*, *C6454* da *TI*, que é capaz de receber e processar os sinais recolhidos de cada eléctrodo a 50kHz e enviar dados para o computador através de pedidos feitos por este. O objectivo posterior deste sistema é fazer o controlo em malha fechada das redes neuronais monitorizadas.

Inicialmente, para desenvolver os algoritmos de detecção começámos por criar um sistema que emula a aquisição de dados através de um outro *DSP*, *c6748* da *TI* que se encontra presente no *LCDK6748*, devido à pouca disponibilidade do sistema de aquisição. Este sistema de emulação revelou-se bastante importante no decurso do projecto, tendo permitido desenvolver e validar todos os algoritmos e fazer a base da *user interface*.

Conseguimos, depois, implementar e validar os algoritmos feitos através da comparação das detecções feitas pelo *DSP*, presente no *MEA2100*, e pelo *software* da *MCS*, *Experimenter Suite*, seguido da implementação da comunicação entre o *DSP* e *PC* usando a técnica de *Polling*, onde o *PC* funciona como *master* e o *DSP* como *slave*.

Os resultados obtidos foram bastante bons, tendo-se conseguido implementar o sistema proposto, monitorizando pelo menos até 128 eléctrodos em tempo real, de modo a que posteriormente sejamos capazes de fazer o controlo da rede neuronal em malha fechada. Conseguimos

também enviar os instantes em que se dão todas estas detecções e gravá-los em ficheiros de texto para posterior análise. Através da *user interface* desenvolvida conseguimos definir que eléctrodos pretendemos monitorizar e quais os parâmetros requeridos para os algoritmos de detecção. Chegámos à conclusão que o relógio implementado pode sofrer atrasos em certas situações, quando existe uma elevada actividade em muitos canais em simultâneo (p.e. 96 eléctrodos com *spikes* a uma taxa de disparo de 10Hz), tornando as gravações menos precisas, no entanto não é um factor crítico visto que o mais importante era conseguir fazer a detecção em tempo real. Depois de validado e caracterizado, o sistema desenvolvido foi utilizado para monitorizar diferentes tipos de culturas neuronais - hipocampais e *DRGs* - em diferentes tipos de substratos - *MEA single well*, *MEA 6-well*, e *MEA* com microfluídicas. Em nenhuma monitorização de actividade neuronal nos deparámos com o problema supramencionado, relativo aos atrasos no relógio. Tal só aconteceu quando gerámos nós próprios os sinais de modo a percebermos as limitações do sistema, onde se impôs taxas de disparo bastante elevadas em múltiplos eléctrodos em simultâneo, algo que não deverá acontecer nas culturas neuronais reais.

8.1 Trabalho futuro

Ainda existem alguns aspectos a melhorar e a implementar no sistema desenvolvido.

Podemos adicionar mais opções na *user interface*, como a escolha da frequência de corte do filtro utilizado, qual a cadência a que pretendemos receber dados do *DSP*, etc. Podemos ainda melhorar o modo de visualização dos gráficos, quer nos *raster plots*, quer nos *interspike histograms* e adicionar mais tipos de gráficos que possam ser interessantes para a análise.

No que toca ao processamento do *DSP* existem ainda alguns aspectos que podem ser melhorados. Os algoritmos de detecção podem ser revistos de modo a tentar torná-los mais eficientes. Na parte da comunicação existe um problema a ser resolvido, que foi detectado na validação do sistema, que pode ocorrer quando a quantidade de dados a serem enviados para o computador excede o espaço disponível da *mailbox*. Podem também ser exploradas novas estratégias de modo a tornar o relógio mais preciso.

Por fim, já estamos aptos a implementar formas de estimular a rede de neurónios que esteja a ser monitorizada como resposta às detecções feitas, podendo ser escolhida a forma e frequência desses estímulos. Assim, podem posteriormente ser aplicados algoritmos de controlo em malha fechada mais complexos e precisos para obtermos a actividade neuronal que queiramos na rede.

Referências

[1] William C. Hall Ben Hayden Anthony-Samuel LaMantia Richard D. Mooney Michael L. Platt Dale Purves Fan Wang Leonard E. White George J. Augustine, David Fitzpatrick. *Neuroscience, 6th edition*. United States of America, Oxford University Press, sixth edição.

[2] Rebecca Knapp et. al Mustafa M. Husain, A. John Rush Max Fink. Speed of response and remission in major depressive disorder with acute electroconvulsive therapy (ect): A consortium for research in ect (core) report. *The Journal of Clinical Psychiatry*.

[3] J.Fleminger Pamela Taylor. Ect for schizophrenia. *The Lancet*, páginas Vol. 35:1380–1383, Junho 1980.

[4] Raheel Shahid Nutan Atre, Atul R. Continuation and maintenance ect in treatment-resistant bipolar disorder. *The Journal of ECT*, páginas Vol. 19:10–16, Março 2003.

[5] Nandurkar S Marsolais EB. Agarwal S, Kobetic R. Functional electrical stimulation for walking in paraplegia: 17-year follow-up of 2 cases. *J. Spinal Cord Med.*, página Vol. 26:86–91, 2003.

[6] Mulsant B. et al. Prudic J., Haskett R.F. Resistance to antidepressant medications and short-term clinical response to ect. *The American Journal of Psychiatry*, Agosto 1996.

[7] Joan Gomez. Subjective side-effects of ect. *The Britsh Journal of Psychiatry*, Janeiro 2018.

[8] Anthony P. Morrison Richard P. Bentall. More harm than good: The case against usingantipsychotic drugs to prevent severe mentalillness. *Journal of Mental Health*, 2002.

[9] Bonnie Schell Brian H. McCorkle W. Thomas Summerfelt Peter J. Weiden Susan M. Essock Nancy H. Covell, Ellen M. Weissman. Distress with medication side effects among persons with severemental illness. *Springer Science*, Julho 2007.

[10] A. H. V. Schapira. Progress in neuroprotection in parkinson's disease. *european journal of neurology*, Março 2008.

[11] Mario Fioravanti Luciano Belloi Orazio Zanetti Luc P. De Vreese, Mirco Neri. Memory rehabilitation in alzheimer's disease: a review of progress. *International Journal of Geriatric Psychiatry*, Agosto 2011.

[12] AmyBrooks-Kayal et al. Margaret P.Jacobs, Gabrielle G.Leblanc. Curing epilepsy: Progress and future directions. *Epilepsy Behavior*, páginas Vol. 14:438–445, Março 2009.

[13] J. C. Rothwell P. D. Thompson P. A. Merton C. D. Marsden Y. Ugawa, B. L. Day. Modulation of motor cortical excitability by electrical stimulation over the cerebellum in man. *The Journal of Physiology*, Setembro 1991.

[14] Alim LouisBenabid. Deep brain stimulation for parkinson's disease. *Current Opinion in Neurobiology*, páginas Vol. 13:696–706, Dezembro 2003.

[15] W. Harrison X. Sun F. G. Zeng, S. Rebscher e H. Feng. Cochlear implants: System design, integration, and evaluation. *IEEE Reviews in Biomedical Engineering*, páginas Vol. 1:115–142, 2008.

[16] Multichannel systems. Mea2100 user guide. Relatório técnico.

[17] Mary Summo Maida David L. Felten, M. Kerry O'Banion. *NETTER'S ATLAS OF NEUROSCIENCE, THIRD EDITION*. Elsevier, sixth edição.

[18] Ahtiainen A. Hyttinen J.A.K. Tanskanen, J.M.A. Extracellular electrical stimulation-based in vitro neuroscience: A minireview of methods and a paradigm shift proposal. *2019 26th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2019*, páginas 883–886, Junho 2019.

[19] Satoshi Ayuzawa et al. Sumire Ishiyama, Yasushi Shibata. Clinical effect of c2 peripheral nerve field stimulation using electroacupuncture for primary headache. *Neuromodulation*, páginas Vol. 21:793–796, Março 2018.

[20] B. Kolb e I. Q. Whishaw. Fundamentals of human neuropsychology. *New York, NY, USA: Worth Publishers*, 2015.

[21] F. M. Gafarov. Neural electrical activity and neural network growth. *Neural Networks*, páginas vol. 101, pp. 15–24, Maio 2018.

[22] M.P. OSBORNE L.H. FINLAYSON. Secretory activity of neurons and related electrical activity. *Advances in Comparative Physiology and Biochemistry*, páginas Vol. 6:165–258, 1975.

[23] Bruce P. Bean. The action potential in mammalian central neurons. *nature reviews neuroscience*, página Vol. 8:451–465, Junho 2007.

[24] Francesco Difato et al. Aurel Vasile Martiniuc, Dirk Saalfrank. Tracking the evolution of neural network activity in uninterrupted long-term mea recordings. *7th Int. Meeting on Substrate-Integrated Microelectrodes*, 2010.

[25] H. Jung et al. R. kim, S. Joo. Recent trends in microelectrode array technology for in vitro neural interface platform. *Biomedical Engineering Letters*, páginas 129–141, 2014.

[26] M. Taketani e Eds. M. Baudry. Advances in network electrophysiol- ogy using multi-electrode arrays. *Boston, MA, USA: Springer*, 2006.

[27] H. H. Yoon et al. H. Cho, Y. K. Seo. Neural stimulation on human bone marrow-derived mesenchymal stem cells by extremely low frequency electromagnetic fields. *Biotechnol. Prog., vol. 28, pp. 1329–1335, July 2012*.

[28] et al. Kim Quasthoff, Stefano Ferrea. Freshly frozen e18 rat cortical cells can generate functional neural networks after standard cryopreservation and thawing procedures. *Cytotechnology*, 2015.

[29] Jeffrey Kramer Quinn H. Hogan Andrew S. Koopmeiners, Samantha Mueller. Effect of electrical field stimulation on dorsal root ganglion neuronal function. *Neuromodulation*, Fevereiro 2013.

[30] et al. Kim Quasthoff, Stefano Ferrea. A cmos-based microelectrode array for interaction with neuronal cultures. *Journal of Neuroscience Methods*, 2007.

[31] H. Hazan e et al. Ziv. Closed loop experiment manager (clem)—an open and inexpensive solution for multichannel electrophysiological recordings and closed loop experiments. *Frontiers in Neuroscience*, 2017.

[32] Daniel A.Wagenaar. Effective parameters for stimulation of dissociated cultures using multi-electrode arrays. *Journal of neuroscience methods*, 2004.

[33] Luca Citi Luigi Raffo Danilo Pani, Francesco Usai. Real-time processing of tflife neural signals on embedded dsp platforms: A case study. *5th International IEEE/EMBS Conference on Neural Engineering*, páginas Vol. 21:793–796, 2011.

[34] Jonathan Landes et al. Peng Cong, Piyush Karande. A 32-channel modular bi-directional neural interface system with embedded dsp for closed-loop operation. *40th European Solid State Circuits Conference (ESSCIRC)*, 2014.

[35] E. A. Tomasella D. Baschirotto A. Vassanelli S. Maschietto M. De Matteis M. Tambaro, M. Vallicelli. A 10 msample/sec digital neural spike detection for a 1024 pixels multi transistor array sensor. *2019 26th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2019*, páginas 711–714, 2019.

[36] Bernabe Serrano-Gotarredona Teresa Ahmadi-Farsani, Javad Linares-Barranco. Digital-signal-processor realization of izhikevich neural network for real-time interaction with electrophysiology experiments. *2019 26th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2019*, páginas 899–902, 2019.

[37] Grace Mathew Amrutur-Bharadwaj Sikdar Sujit Kumar George, Jude Baby Abraham. Robot navigation using neuro-electronic hybrid systems. *Proceedings of the IEEE International Conference on VLSI Design*, páginas 93–98, 2015.

[38] Marc W. Slutzky et al. Max O. Krucoff, Shervin Rahimpour. Enhancing nervous system recovery through neurobiologics, neural interface training, and neurorehabilitation. *frontiers in Neuroscience*, páginas Vol. 1:115–142, Dezembro.

[39] et al. J.R. Wolpaw. Brain–computer interfaces for communication and control. *Clin. Neurophysiol.*, páginas Vol. 113:767–791.

[40] Miguel A.L.Nicolelis Mikhail A.Lebedev. Brain–machine interfaces: past, present and future. *Arch. Phys. Med. Rehabil.*, páginas Vol. 29:536–546.

[41] Timothy L et al. Mikhail A. Lebedev, Andrew J. Tate. Future developments in brain-machine interface research. *Clinics*, 2011.

[42] et al. Rodrigo A. Brant Fernandes, Bruno Diniz. Artificial vision through neuronal stimulation. *Neuroscience Letters*, páginas Vol. 519:122–128.

[43] Moxon K. Markowitz R. et al. Chapin, J. Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex. *Nat Neurosci*, página Vol. 2:664–670.

[44] Stambaugh C. Kralik J. et al. Wessberg, J. Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature*, página Vol. 408:361–365.

[45] Multichannel systems. Mea2100-system manual. Relatório técnico.

[46] Multichannel systems. 256mea for usb-mea256-system and mea2100-256-systems. Relatório técnico.

[47] Texas Instruments. Introduction to tms320c6000 dsp optimization. Relatório técnico, 2011.

[48] Texas Instruments. Tms320c6454. Relatório técnico, 2012.

[49] Multichannel systems. Mea2100-256 user guide. Relatório técnico.

[50] Texas Intruments. Edma3 driver user guide. Relatório técnico, 2014.

[51] Texas Intruments. Tms320c6748tm fixed- and floating-point dsp. Relatório técnico, 2017.

[52] Texas Intruments. Omap-l138/c6748 low-cost development kit (lcdk) user's guide. Relatório técnico, 2019.

[53] Mateus J.C. Lopes C.D.F. et al Heiney, K. µspikehunter: An advanced computational tool for the analysis of neuronal communication and action potential propagation in microfuidic platforms. *Sci Rep*, Abril 2019.