FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
INSTITUTE OF INFORMATICS
COMPUTER ENGINEERING

JOÃO FELIPE LEIDENS

Development of a Polygraph Graphical User Interface for Acquisition of Physiological Signals

Computer Engineering Graduation Work.

Prof. Dr. Renata Galante
Advisor

Porto Alegre
2014

RESUMO ESTENDIDO

1 INTRODUÇÃO

O objetivo desta seção é apresentar um resumo estendido em português do trabalho originalmente escrito em inglês. Esta seção fornece todas as noções básicas necessárias para a compreensão simples do do contexto do trabalho, seus objetivos e, finalmente, seu desenvolvimento.

1.1 Contexto

Este trabalho faz parte de um projeto com vários parceiros chamado INTENSE. Este projeto é dirigido principalmente pelo laboratório LE2S (pertencente ao CEA, onde o trabalho aqui apresentado foi desenvolvido) e pela sociedade SORIM C.R.M. O principal objetivo deste projeto consiste em desenvolver um dispositivo médico capaz de tratar certas patologias cardíacas via estimulação da atividade elétrica do nervo vago em nível cervical. A atividade gerada pelo nervo vago é registrada como o eletroneurograma, que consiste em um sinal capturado por um grupo de sensores colocados na superfície de um eletrodo cilíndrico (chamado eletrodo cuff).

Este eletrodo é cirurgicamente inserido em volta do nervo vago do paciente a nível cervical; os dados recuperados são então submetidos a análises matemáticas a fim de descobrir alguma informação cardíaca. Uma vez que tratamentos farmacológicos não surtem efeito em alguns pacientes, uma solução possível para tratar algumas destas patologias é através da estimulação do nervo vago. Uma outra aplicação, por exemplo, é substituir o uso de marca passos por um eletrodo cuff, gerando o processo de estimulação. Para alcançar esse objetivo, o sistema chamado NeuroPXI foi desenvolvido no laboratório LE2S. Este sistema pode aplicar diversos padrões de estimulação no paciente e também registrar diversos sinais conectando equipamentos médicos e sensores a seus canais, processo esse que se chama modo de aquisição. Durante alguns experimentos prévios, esses eletrodos foram cirurgicamente implantados em porcos, uma vez que a estrutura do nervo vago deste animal é muito similar a dos humanos. Esses experimentos mostraram ótimos resultados em relação ao funcionamento do sistema NeuroPXI para coletar os sinais fisiológicos. Assim, uma maneira adequada de visualizar esse sinais é necessária

1.2 Motivação

O maior problema consiste no grande volume de dados a serem trabalhados que são capturados pelo sistema NeuroPXI. Considerando que ele funciona com uma frequência de aquisição de 20KHz, cada ponto de medida é armazenado ocupando 2 bytes e que o sistema é previsto para ter até 1024 canais, isso resulta em um taxa de dados de até 40Mbytes por segundo a ser tratada. Embora esse fluxo seja perfeitamente tolerável para o barramento PXI, que suporta até 100Mbytes/s, o software responsável pelo registro deve conseguir lidar com esse volume de dados. Um problema adicional é que, mesmo que um baixo número de canais esteja sendo usado, NeuroPXI envia páginas de dados de todos os canais mesmo assim. Isso significa que os dados úteis devem ser minerados dos dispensáveis e a exibição e o registro dos sinais deve ser mais rápida do que o intervalo em que cada página é recebida do barramento via DMA. Sistemas e software similares podem ser encontrados no mercado; mas eles apresentam dois problemas que os impossibilitam de serem utilizados no projeto INTENSE: o sistema de aquisição não trabalha com uma frequência tão alta e portanto é considerado lento para o projeto; o software disponível obviamente não é compátivel com o sistema NeuroPXI e nem é open source. Portanto, tornam-se dispensáveis para o projeto.

1.3 Objetivo

O desenvolvimento de um software para esse sistema é então proposto. Chamado de Intense, ele é o foco deste trabalho (o software leva o nome em letras minúsculas enquanto que o projeto no qual está inserido leva o nome em letras maiúsculas). Intense tem o objetivo de fornecer uma maneira própria e fácil de visualizar os sinais capturados pelo sistema NeuroPXI e armazená-los no disco usando um formato apropriado para serem submetidos a análises matemáticas posteriores. Para suportar a crescente demanda de informação que precisa ser registrada, o software Intense necessita ser expandível, permitindo que novos sinais sejam adicionados futuramente e que possíveis mudanças no hardware possam ser levadas em conta.

2 TRABALHO

Nesta seção são mostradas caracterísitcas referentes ao desenvolvimento do software, entre elas as tecnologias e ferramentas incorporadas e as abordagens utilizadas.

2.1 Proposta

A proposta do software chamado Intense é portanto elaborar uma interface gráfica do usuário que possibilite a configuração do sistema NeuroPXI, a aquisição das páginas de dados oriundas do sistema via barramento PXI utilizando DMA, a criação dos chamados experimentos e dos sinais conforme o usuário desejar, a plotagem em tempo real dos gráficos dos sinais e, finalmente, o armazenamento dos dados adquiridos (sinais) em formato SMR (formato utilizado pelo software Spike2). Comparando a exibição dos gráficos referentes aos sinais em tempo real com o armazenamento dos mesmos, a prioridade deve sempre ser dada para o armazenamento, uma vez que são os dados armazenados que serão submetidos a análises. A exibição é uma ferramenta utilizada na hora do experimento para certificar que a suíte de aquisição está funcionando propriamente e todos os dados desejados estão sendo coletados.

As principais características de um experimento são: nome do experimento, junto com a opção de concatenar a hora e a data ao nome nos arquivos gerados; a duração do experimento; opção de particionar os arquivos gerados (arquivos SMR) em segmentos menores com uma duração definida em minutos; diretório onde os arquivos devem ser salvos; opção de usar o filtro DC implementado na FPGA; opção de utilizar uma sub frequência para exibição dos sinais em tempo real; opção de quais dos sinais criados utilizar no experimento; opção de utilizar e configurar a função trigger;

Um experimento pode ter até 256 sinais. O usuário deve poder criar e gerenciar sinais facilmente. As principais características de um sinal são: nome do sinal e comentário; opção de armazenar o sinal (arquivo SMR); opção de plotar o gráfico do sinal em tempo real; configurar o ganho ASIC proporcionado pelo sistema; configuração do ganho externo conectado ao sinal; opção de utilizar a função trigger para esse sinal; unidade do sinal; cor do sinal; canal associado ao sinal. Uma vez criado um experimento e seus devidos sinais os mesmos devem ser salvos automaticamente utilizando um formato de arquivo adequado, possibilitando modificações posteriormente.

2.1 Desenvolvimento

O desenvolvimento do software contou com o uso de diversas ferramentas e tecnologias: a linguagem de programção C++, graças ao seu grande uso nesse tipo de software devido a sua rapidez e flexibilidade, considerando também sua compatibilidade com a linguagem C, uma vez que o acesso ao sistema NeuroPXI foi projetado usando C; biblioteca Qt, devido a sua facilidade de criar interfaces gráficas de alta qualidade e fácil reaproveitamento de classes e Widgets; biblioteca Boost, graças ao seu enorme número de implementações e containers prontos para uso, além de ser uma biblioteca de alta qualidade sendo usada inclusive na normalização das versões do C++; biblioteca TinyXML, uma vez que proporciona uma fácil manipulação de arquivos XML e um tamanho extremamente reduzido comparado a outras bibliotecas que manipulam XML.

O ambiente de trabalho consiste em uma SDE criada no próprio laboratório onde o trabalho foi desenvolvido; para realizar o versionamento, a ferramenta TortoiseSVN foi utilizada. A partir da SDE é possível gerar projetos para o Microsoft Visual Studio 2010, que é o ambiente de programação utilizado, além do QtDesigner para criar alguns dos widgets usados no software.

Para dar prioridade para o armazenamento dos dados (considerada mais importante) sobre a exibição em tempo real dos mesmos, a função de callback chamada cada vez que uma página de dados esta disponível via DMA insere esses dados em um container implementado como fila destinado aos dados para o armazenamento. Essa implementação de fila faz uso de semáforos para sincronização. Assim, somente após os dados da fila de armazenamento serem consumidos esse mesmos dados se tornam disponíveis e inseridos na fila de exibição, onde a thread que trata da exibição dos gráficos em tempo real consome os dados dessa fila. Para a criação dessas threads e implementações de buffers foi usada a biblioteca Boost.

A criação de experimentos e sinais exigia um formato adequado para serem salvos. Por isso foi utilizado o formato XML, que além de prover as funções necessárias também é legível, o que significa que um experimento pode ser facilmente modificado a mão e aberto com o software Intense. Com o uso da biblioteca TinyXML, um parser completo foi criado; os arquivos XML com os experimentos e sinais são gerados e podem ser modificados facilmente. Ao editar ou abrir um experimento usando Intense, o parser realiza a checagem para verificar se os dados contidos no arquivos XML estão

corretos; no caso que uma modificação a mão considerada errada tenha sido feita, é informado no console do Intense o problema encontrado e a linha correspondente no arquivo para o usuário realizar a correção.

A divisão do software foi feita em várias bibliotecas que posteriormente geram um arquivo DLL na versão compilada. A principal divisão para o software Intense foi em: classes implementadas para a aquisição de páginas via DMA, configuração do sistema NeuroPXI, parser XML e implementação de fila para inserção das páginas; e outra para as classes que implementam a interface gráfica, contendo todas as janelas e widgets criados. Essa modularidade permite a separação do núcleo que contém toda a parte funcional padrão relacionada ao sistema NeuroPXI dos detalhes peculiares que fazem parte somente da interface gráfica criada. Assim, caso seja desejado criar uma nova interface gráfica, toda a implementação base não precisará ser modificada. Isso permite a elaboração de um software modular, de maior qualidade e entendimento.

3 ESTUDO DE CASO

Uma vez que o software encontrava-se praticamente pronto, foi realizado um experimento utilizando uma minhoca da terra como espécime. A minhoca é um bom modelo in vivo para testar os eletrodos cuff e o registro dos sinais de eletroneurograma. Nesse experimento a minhoca é anestesiada e colocada em uma gaiola de Faraday, evitando assim interferência de sinais externos. O amplificador é configurado com um ganho de mil vezes e, juntamente com o ganho do pré-amplificador, resulta em um ganho de dez mil vezes.

Para esse experimento, havia o interesse particular de estimular a minhoca e observar as respostas geradas. Sinais externos foram conectados ao sistema NeuroPXI
a verificar o funcionamento correto da suíte de aquisição. A tensão RMS ao redor do
trodo cuff é então medida, bem como a corrente RMS em um resistor colocado em série com o eletrodo cuff.

Depois de todas as medidas e calibragens serem feitas, o experimento está pronto para começar. Um novo experimento foi criado usando Intense, onde foram criados dois sinais de eletroneurograma e um sinal para marcar o momento da estimulação. A cada vez que uma estimulação é feita, os sinais de eletroneurograma são atualizados. O experimento foi bem sucedido e as respostas puderam ser claramente vistas na janela de exibição dos sinais em tempo real do Intense. Após o experimento,

os arquivos SMR gerados foram coletados para serem submetidos a análises pelos biólogos e engenheiros usando Spike2 e MATLAB.

4 CONCLUSÕES

Um ponto peculiar do projeto ao qual o software Intense está inserido é o fato de que muitos domínios de conhecimento diferentes convergem. Pode-se verificar que a tecnologia emcompassa diferentes maneiras de identificar a natureza de certas condições, permitindo intervenções com o objetivo de aumentar a expectativa e a qualidade de vida; sendo pelo uso de novos dispositivos, métodos farmacológicos ou biológicos. O estudo de caso mostra que o software Intense está pronto para ser usado e é estável para exercer as tarefas a ele designadas. Mesmo ele estando inserido em um projeto de longo termo, as chances de o conjunto todo vir a ser uma versão comercial ou colaborar com algum avanço na área médica é promissora.

Obviamente há espaço para melhorias. O tempo curto para o desenvolvimento do software permitiu criar uma versão totalmente funcional, mas sempre há a possibilidade de adicionar novas funções. Futuras modificações foram previstas e provavelmente ocorrerão onforme o andamento do projeto como um todo.

Embora Intense tenha sido concebido especificamente para trabalhar com o sistema NeuroPXI, seus conceitos núcleo e ideias devem ser facilmente transferidos para qualquer aplicação que tenha que lidar com quantidades de dados com características similares aos usados aqui e seu sistema de plotagem pode ser usado para qualquer aplicação que necessite traçar gráficos em tempo real. Sua divisão modular permite futuras extensões e reusos; o que significa que as classes responsáveis por exibir e registrar os dados podem ser aplicadas em qualquer outro problema que faça as mesmas exigências.

ABSTRACT

The focus of this work lies on the development of software designed to interface with a medical device that captures physiological signals. Thus, the software must be capable of handling high data rates, providing a real time display of all the information on the screen and storing this information using a suitable format for later analysis. This work was performed at CEA (Commissariat à l'énergie atomique et aux énergies alternatives, France) during a time period of six months. The project in which this work is inserted is driven by SORIN C.R.M. and it has as its main goal the development of new implantable electronic devices destined to apply the use of neurostimulation in pathologies related to the cardiac insufficiency.

The laboratory LE2S, where the medical device and the software here presented were developed, is concerned with demonstrating that it is possible to find cardiac information in signals that are captured on the periphery of the vagus nerve (the name electroneurogram is attributed to this specific signal). To take advantage of the information that this signal carries, which has very weak amplitude (some microvolts), it is necessary to combine other signals along with the electroneurogram (e.g. electrocardiogram, electromyogram and arterial pressure).

Therefore, the laboratory needed software capable of configuring the device responsible for collecting these signals (this device is often referred as NeuroPXI system, or sometimes BioMEA, which is the previous version of the device), storing all the information acquired during an experiment and displaying it on the screen.

Keywords: acquisition, signals, storage, display, NeuroPXI, medical.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AGNES	Asic for General Neurons Electrical Study
CEA	Commissariat à l'Énergie Atomique et aux Énergies Alternatives
DMA	Direct memory access
GUI	Graphical User Interface
INCIA	Institut de Neurosciences Cognitives et Intégratives d'Aquitaine
MEA	Multielectrode arrays
PCI	Peripheral Component Interconnect
PXI	PCI eXtensions for Instrumentation
SDE	Integrated Development Environment
SPB	Signal Processing Board
UML	Unified Modeling Language
USB	Universal Serial Bus
XML	Extensible Markup Language

SUMMARY

1 INTRODUCTION

The goal of INTENSE, a multi-partner project developed mainly by LE2S and SORIN C.R.M., consists of developing a medical device capable of treating some certain cardiac pathologies by stimulating and recording the vagus nerve at the cervical level. The activity generated in the vagus nerve is registered as the electroneurogram, which is a signal captured by a set of sensors placed on the surface of a cuff electrode.

This electrode is surgically inserted around the subject's vagus nerve at the cervical level; the retrieved data is then submitted to mathematical analysis in order to find some cardiac information. Since some pharmacological treatments are unresponsive for some certain patients, one possible solution for treating some cardiac problems is through the stimulation of the vagus nerve. One application, for example, is replacing the use of a pacemaker for one cuff electrodes, generating the necessary stimulation. To achieve this goal, the NeuroPXI system (former BioMEA) developed at LE2S has been improved. This system can apply a number of stimulation patterns to the subject and it can also capture many signals by connecting medical devices and sensors to its channels, which consists of what we call an acquisition mode. During previous experiments, these electrodes were surgically implanted in pigs since the structure of the vagus nerve of this animal is incredibly similar to the human's one, having almost the same thickness. These experiments showed great results concerning the functioning of the NeuroPXI system when gathering signals. Thus, an adequate way of visualizing them must be provided.

The main problem consists of the great amount of data that is captured per second by the NeuroPXI system. Consider that it works at a frequency of 20KHz, each point of measurement is stored using a short variable (which consists of 2 bytes) and that the system is designed to have up to 1024 channels; this results in a data rate of 40Mbytes/s. Although this flux is perfectly handled by the PXI bus, the software must be able to deal with all this data properly. The problem, however, is that even if only a small number of channels are being used; NeuroPXI will send data pages containing information concerning all channels nonetheless. Since the software captures one data page each 25.55ms, the algorithms to separate the desired data from the useless one and to display and store this data must be faster than this rate. Similar software and systems can be found; but they present two main problems: the systems that exist and can be purchased are not fast enough for this project, which means that their acquisition

frequency is lower than NeuroPXI's. This implies the second problem which is the software. The software available on the market was not conceived using the same format as NeuroPXI's and they are not open source; therefore, it becomes useless for this project.

The software called Intense (which takes the same name as the project) is the focus of this work. It intends to deliver a proper and easy-to-use way to visualize and store these signals using the libraries that have already been created in order to have access to the lower layers of the hardware program. The software must store all data obtained in a suitable file format, in order to allow this data to be accessed later on. To support the increasing demand of information that needs to be registered, the Intense GUI must also be expandable, allowing new signals to be added and future hardware modifications to be taken into account. Although Intense was developed specifically to work with NeuroPXI, its core concepts and ideas can be easily transferred to any application that must deal with great amount and similar data. Its division in many libraries makes it modular enough for future extensions and reuse; which mean that the classes responsible for displaying data and storing it could be easily applied to any other application that must deal with the same challenges.

The division of this work in chapters is such that aims for a clear explanation of the project. The Chapter 2 involves the basic concepts that are necessary to understand the project context as a whole and it tries to be as brief as possible and demystify the main properties of the NeuroPXI system. Since it concerns the hardware part of the project, some information about the system had to be concealed due to industrial property. In this part the main signals that the system should be able to acquire are presented and the page structure used to transfer data via the PXI bus. Then an analysis of similar system and software is presented to justify the development of a whole new one. The work proposal is presented next, which contains the information about the functionalities that the software needs to have and some strategies to achieve them. The concept part shows how the software was designed, its core ideas and mechanisms and its validation. With the software ready it is presented a study case involving a living being attached to the NeuroPXI system to test the software. Finally some conclusions concerning the development of such software are sketched in Chapter 7.

2 BASIC CONCEPTS

In this chapter the basic concepts that allow a fully understanding of the project's context are presented. First, the BioMEA and NeuroPXI systems' architecture and characteristics are showed, followed by some of the most common signals that it is able to acquire. The mechanism that the device uses to transfer data through the PXI bus is also presented, as well as a conversion table inherited from the system that will be used later on the development of the software.

2.1 The BioMEA system

BioMEA was the first in vitro device developed at CEA to measure and stimulate a neural network applied to a MEA. This system allows the acquisition and stimulation of a nervous tissue; by doing that, the scientists could better understand how the activation of physiological signals is generated and how these tissues respond to different electrical stimulations.

Figure 2.1 - The BioMEA system

The functioning of BioMEA relies on four AGNES with a MEA attached to them. The nervous tissue that is being studied is placed on the surface of this matrix that will capture all data required. BioMEA used an USB interface to communicate with the PC. The diagram presented below shows a glance at the system architecture.

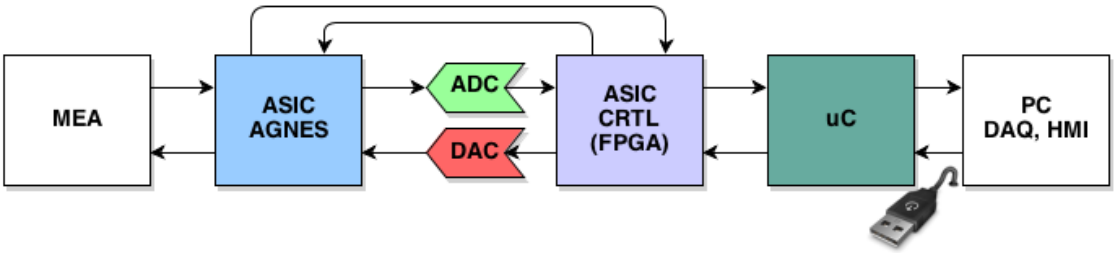

Figure 2.2 - BioMEA's architecture

The project was developed in partnership with INCIA, located in Bordeaux and the technology was later transferred to them. Two papers were published about BioMEA; they can be found in the references section.

2.2 The NeuroPXI system

NeuroPXI is the new system developed at CEA and its main objective presented here is to register the neurobiological signals captured with aid of some other medical devices and electrodes that are connected to this system. The NeuroPXI system allows the implementation of functions for performing real time signal treatment and an increased number of channels for both the stimulation and acquisition of signals (up to 1024 channels are available for future modifications and new functionalities). Here the main details concerning the acquisition of signals performed by this system are presented, as well as an overall of its architecture.

The system is built using four AGNES (a ASIC developed at CEA) and its communication is based on the PXI bus architecture. PXI is an industrial standard which presents as main advantage its high speed data transfer capabilities, which might be up to 120Mb/s. The NeuroPXI is conceived to address 1024 channels using a sampling frequency of 20 KHz, corresponding to a data flux of 40Mbytes/s, which is perfectly manageable using the PXI standard. The NeuroPXI system is built using a commercial chassis which also provides the system's power alimentation. The published paper that describes the NeuroPXI system more in detail can be found in the references section.

Figure 2.3 - PXIS2506 chassis and PCI-PXI Bridges. PXI-8565 (left) and PCIe-8560 (right)

The current system's configuration has 256 channels and contains the four AGNES implemented on the AFE256 board. This board contains all analogical components required, including the analog/digital and digital/analog converters. A 64-channel control module is built around each AGNES and implemented on the FPGA.

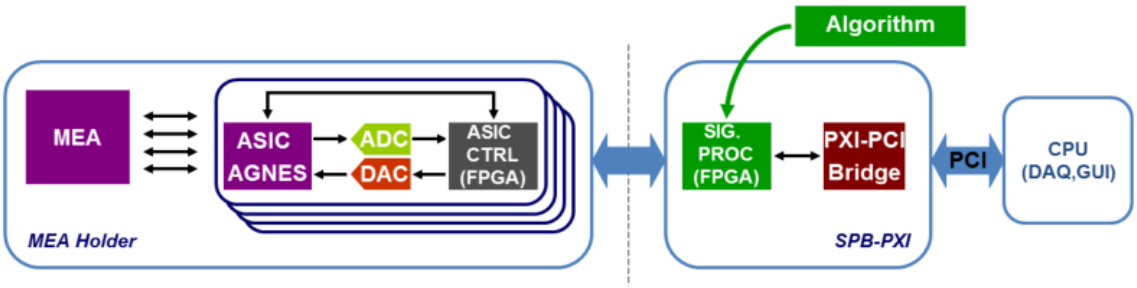

Figure 2.4 - NeuroPXI's architeture

The NeuroPXI is composed by a number of boards developed at CEA being the SPB-PXI3U board at the core of the NeuroPXI system. Its architecture is presented below. The FPGAs shown in grey are responsible for decoding the signals of the local bus and generating the test interface. The FPGA (name omitted due to industrial interests) shown in green integrates the NeuroPXI's registers; the hardware control modules and the real time signal treatment modules as well.

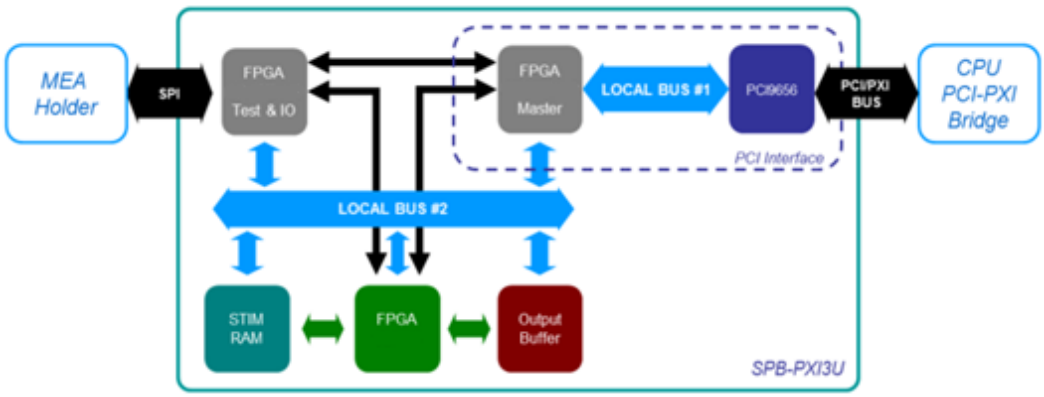

Figure 2.5 - Architecture of SPB-PXI3U

The data transfer is performed through data pages. In the NeuroPXI system, the registers might be accessed in DMA mode, which provides high speed transfers in blocks. The data transfer may be in both ways:

Data reading for acquisition (SPB-PXI3U → PC)

Patterns' transfer for stimulation (PC → SPB-PXI3U)

For the Intense GUI we will be concerned with the data pages retrieved via DMA. The image below shows a simplified scheme of the process used for generating data pages and gathering them.

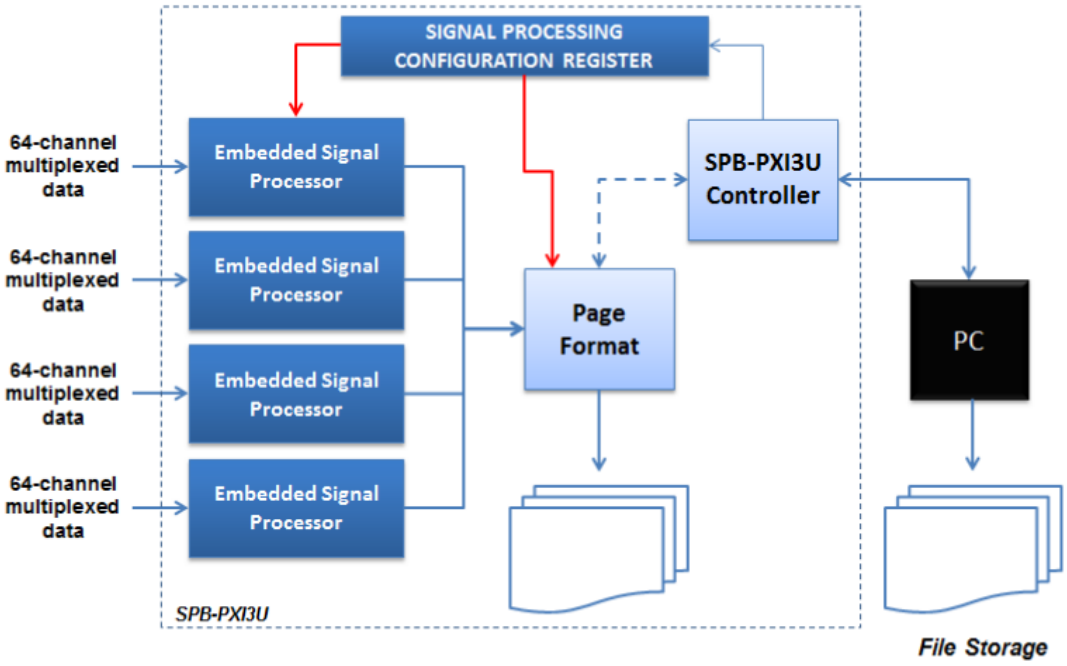

Figure 2.6 - Data storage

The interval time between each page to be received is 25.55ms, where each page contains 511 samples for each channel ($511/25.55\text{ms} = 20\text{KHz}$). Each one of these samples consists of a short data type (2 bytes, 16 bits). Concerning the GUI, after receiving a data page, these samples must be converted to their corresponding value in voltage, according to the table shown below:

Vin Bits	Sign bit	ADC Bits														
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2.5V	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2.5V-1LSB	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0
+1 LSB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1 LSB	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-2.5V+1LSB	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
-2,5V	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.1 – Bit to volt conversion table

2.3 Signals to acquire

The list of signals that can be acquired using the NeuroPXI system is vast and customizable. Therefore, here are presented the most important signals to be acquired taking into account the INTENSE project.

- **Electroneurogram (ENG and ENGi)**

The electroneurogram is the most important signal to acquire. In order to measure this signal, a cuff electrode is placed around the vagus nerve. The currently used electrodes contain four contact surfaces. This means that up to four ENG signals may be acquired. A pointed mono fiber electrode might also be used for this purpose; however, in this case the procedure is invasive and not used in humans. This type of signal receives the more specific name ENGi.

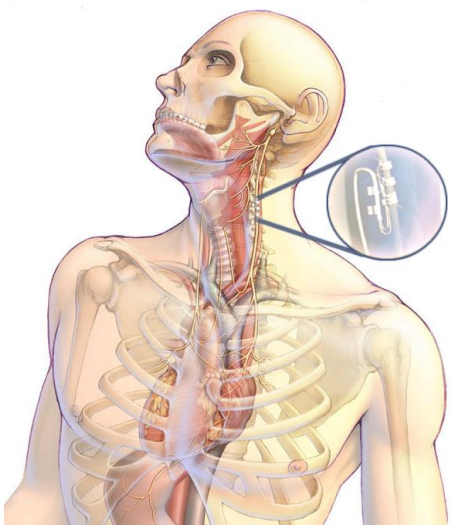

Figure 2.7 – Placement of the cuff electrode around the left vagus nerve

Differently from other signals, the ENG signals are not directly connected to the NeuroPXI system. Firstly, they are connected to a pre amplifier (MCS uPA32) with a gain of 10X. Then, the output of this pre amplifier is connected to an amplifier (PGA 64) with a usual gain of 1000X (programmable device, this value may vary). This signal must be amplified because the surface of the captors placed on the cuff electrode is relatively far from the electrical activity of the nerve. In addition, this activity is isolated by a thin small layer of fat. This leads to a really weak and noisy signal when captured.

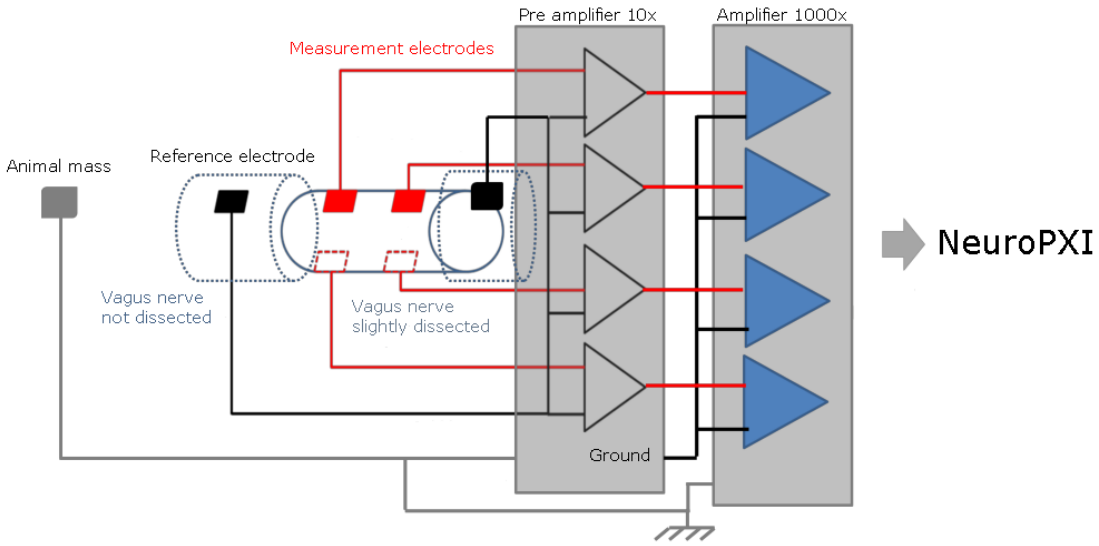

Figure 2.8 - Amplification of the ENG signal

Figure 2.9 - Pre amplifier MCS uPA32 and amplifier PGA 64

For this reason we also capture a list of other signals to be analyzed along with the ENG signals in order to find cardiac information using signal treatment. The two other signals that are mainly used for this purpose is the ECG and the PA, which will be explained later.

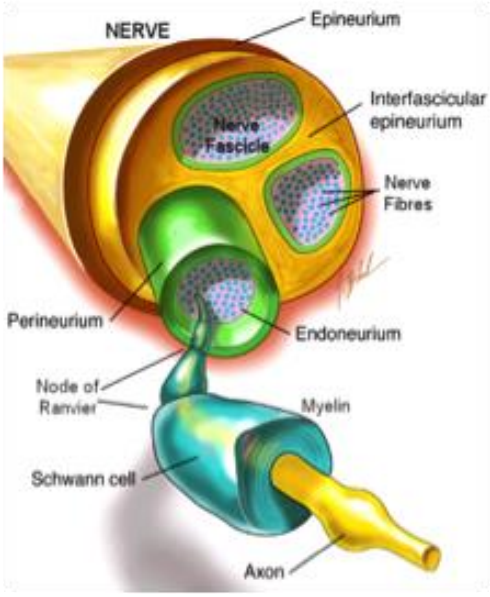

Figure 2.10 - An illustration of a vagus nerve

- **Electrocardiogram (ECG)**

The electrocardiogram is a graphical representation of the electric potential that commands the muscular activity of the heart. This potential is captured using electrodes disposed on the surface of the patient's skin. The image below shows an ECG captured with the NeuroPXI system of an experiment with a pig.

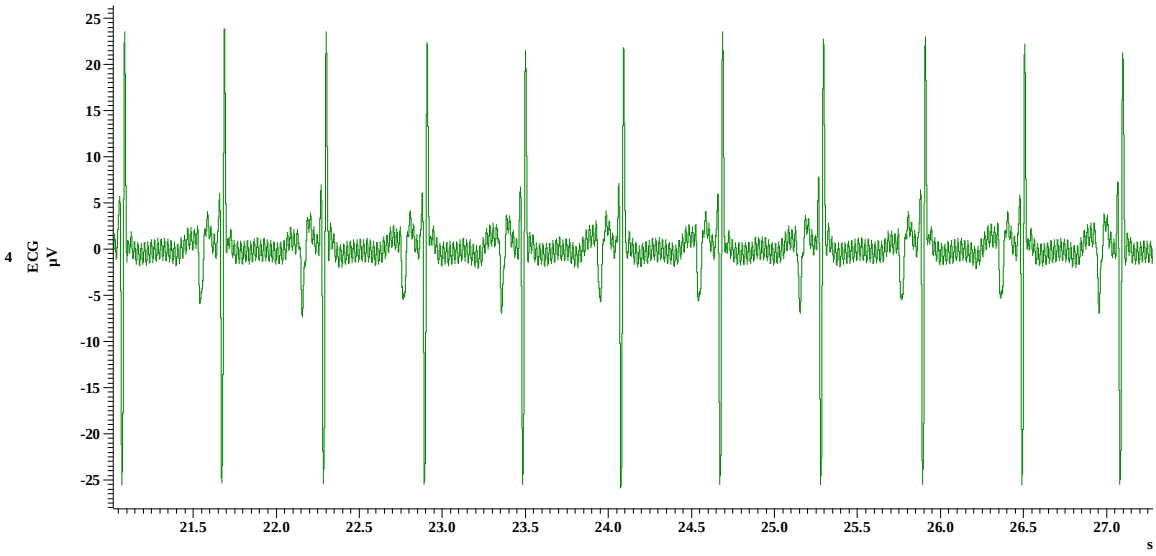

Figure 2.11 - ECG of a pig. Waveform captured with NeuroPXI

- **Electromyogram (EMG)**

The electromyogram is basically the analysis the electrical activity of the muscles during a period of relaxation and activation. This activity is gathered using electrodes located on a thin needle that is placed in the desired muscle. The activity of the superficial muscles, for instance, can be gathered using electrodes placed on the skin.

- **Arterial blood pressure (PA)**

The arterial blood pressure is an invasive technique to monitor the intra-vascular arterial pressure. A carotid catheter is used to measure the arterial pressure for each bloody stream.

- **Capnogram ([CO2])**

The measurement of the amount of carbon dioxide in the expired air gives information about the elimination of this same molecule by the lungs, eventual changes in the dioxide carbon’s production by the tissues and its transport through the circulatory system.

- **Other signals**

  Some other signals can be acquired, among them are:

  - Modulation of the stomach's volume (PINS)
  - Electrical modulation (iStim)

A manual button (MAN) will also be registered and used as a flag to indicate modifications in the respiratory modulation and the cardiovascular modulation. Since these events are triggered by external factor (for example, the cardiovascular modulation may be triggered by the injection of adrenaline in the patient) the button will be used to flag that some event just occurred.

The table shown below shows the main characteristics of the signals presented. These are just the mains signals used; however, it's possible to connect up to 256 signals to the current configuration of the NeuroPXI system.

Signal	Symbol	Sensor	Amplitude
Peripheral electroneurogram	ENG	Cuff electrodes	Some uV
Mono fiber electroneurogram	ENGi	Invasive electrodes	Some uV
Electrocardiogram	ECG	Cutaneous electrodes	mV
Arterial pressure	PA	Carotid catheter	mV
Capnogram	[CO2]	Capnometer	mV
Electromyogram	EMG	Cutaneous electrodes	mV
Respiratory modulation	MAN	Respirator	[0:2.5V]
Stomach's volume modulation	PINS	Stomach balloon	[-2.5V:+2.5V]
Cardiovascular modulation	MAN	Drug injection	[0:2.5V]
Electrical modulation	IStim	Electric stimulator	-

Table 2.2 - Main characteristics of the signals presented

3 RELATED WORKS

Although NeuroPXI is a peculiar system, there are some related systems on the market currently. Here similar software that has some common goals as Intense is presented.

3.1 NeuroPXI v1.12.5.11

This was the first software created for testing BioMEA and then NeuroPXI. This software was created by INCIA, and, when the technology of BioMEA was transferred to them, the source code was no longer available at CEA. In addition, its development was still not complete and therefore many features were missing. These were the two main reasons to start creating a new GUI from the scratch.

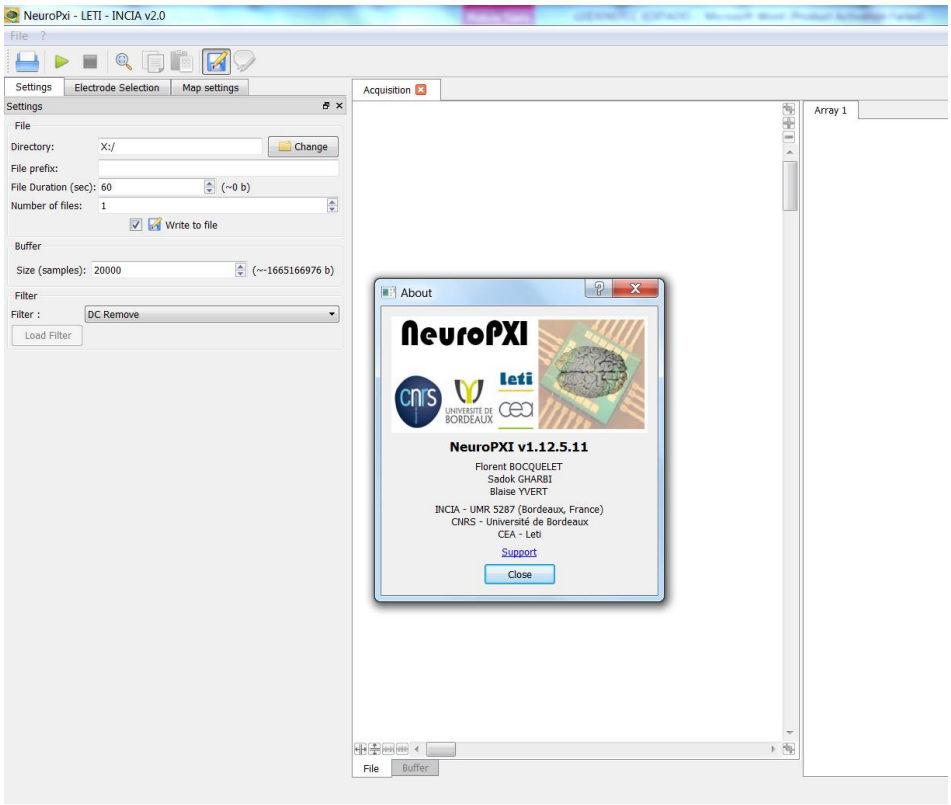

Figure 3.1 - NeuroPXI v1.12.5.11, the first software created for NeuroPXI

3.2 Biopac Acknowledge

Biopac has a similar system called MP150 that is also able to acquire signals. The main reason that the Biopac's system wasn't used by the biologists at CEA was

that, at least at the time that this document was written, the acquiring frequency of this system was not high enough; therefore, it would not serve their needs. NeuroPXI's high frequency of 20KHz, on the other hand, was considered fast enough to find a pathological information by analysing the signals captured.

Figure 3.2 - Biopac MP150 System

Acknowledge is the name of the software that comes with Biopac's system. Since its code is not open source and its structure and data transfer is not the same as NeuroPXI's, the use of this software became useless. The software here presented developed has all the main functions presented in Acknowledge. Some other features that would also have been welcome in Intense are not present due to the tight schedule to develop it. Among these functionalities lies a better way to analyze data and import SMR files to Intense; however, since the biologists will use Spike2 and MATLAB to perform this analysis, this feature should not be missed.

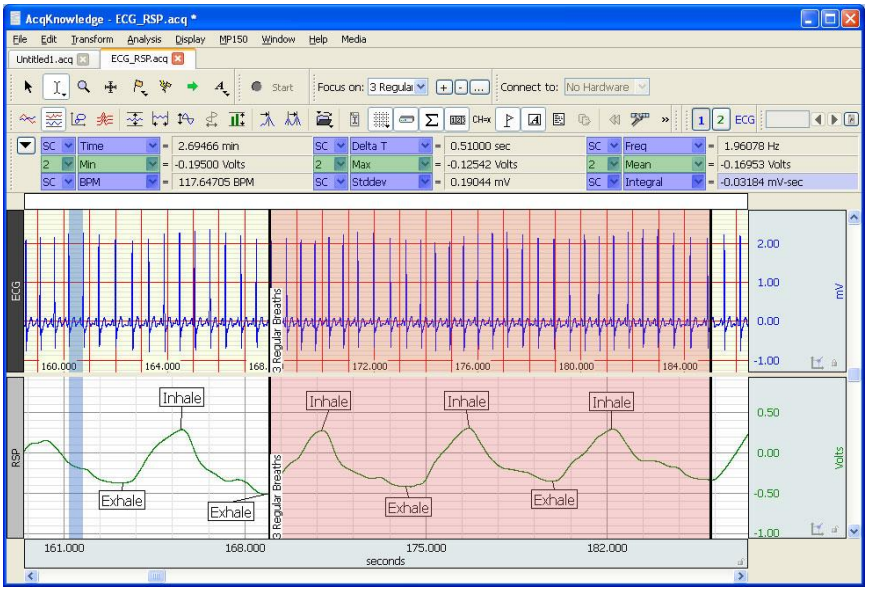

Figure 3.3 - Acknowledge's main window during acquisition

4 WORK PROPOSAL

Here are presented the main characteristics that the software must present and its main goals. The language, framework and assembly of libraries used are also shown here; as well as the diagrams that depict each main function of Intense.

4.1 General view

In order to perform the signal acquisition in an easy way, a proper GUI that allows access to the low level components is necessary. The scope of the work presented here is to develop software capable of capturing the data pages from the NeuroPXI system, displaying them on the screen giving the user an impression o a real time application and storing them on disk using the Son32 file format. The data that is stored will then be submitted to analysis and treatments using MATLAB and Spike2 by engineers and biologists. So it is possible to divide the functions that the software will present as following:

Store data retrieved on the hard disk using the Son32 file format. This is the format used by the software named Spike2. All signals will be acquired by the system with a frequency of 20 KHz. These signals must always be stored using this very same frequency of 20 KHz. This is considered the main task of the GUI, apart from configuring and controlling the NeuroPXI system.

Plot the graphs for each signal that the user desires. All signals are acquired by the system with a frequency of 20 KHz. However, this frequency may be lower than 20 KHz for some signals when plotting the graphs (for performance reasons).

The software was developed using C++ and Qt. This approach allowed a fast performance and access to the libraries that have already been developed (compatibility with C language) by using the C++ language, as well as the conception of a robust and high quality interface by using Qt. Some Boost resources are also used like multithreading and a ring buffer implementation.

Figure 4.1 - Language, assembly of libraries and framework used

Notice that the GUI is destined to the acquisition only; it does not consider the stimulation mode. It's also possible to notice in the image showed below that many APIs have already been developed. By using them, further knowledge of the electronic components is abstracted, since the basic functions implemented in the board can be performed by the use of API calls.

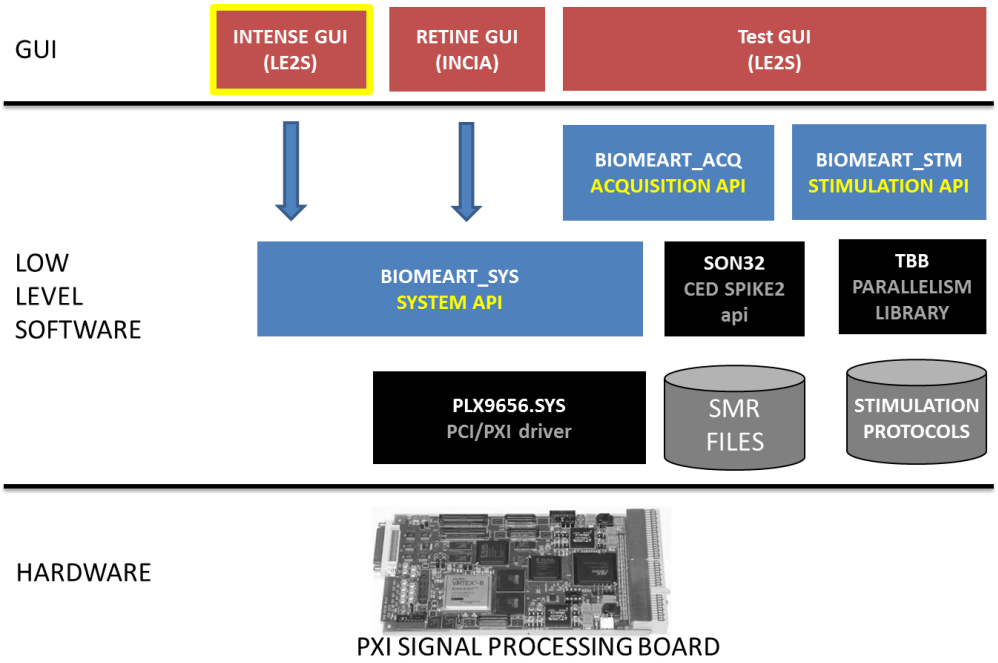

Figure 4.2 - Project hierarchy

4.2 Requirement definition

The first task to develop the software consisted of filling a document with all functionalities that the software needed to have. In order to do that, discussions with engineers involved in the project and tests with other similar software such as AcqKnowledge, by Biopac were of great aid. After gathering all the information it was possible to identify some key elements listed below.

- **Experiment**

It is necessary to model an experiment format. The experiment must contain every parameter regarding the system's general options and its list of signals. The user must also be able to load and modify previous experiments.

- **Setting up signals**

A great variety of signals may be used; ranging from different units, gains provided by the AGNES and natures as a whole. Therefore, a generic way of creating signals must be provided. The user must be able to create up to 256 signals for one experiment (maximum number of channels available at the moment, but it should be expanded in future versions) and manage them as necessary.

- **File manager**

The Son32 files that are generated must be managed automatically by the software, giving them a proper name and creating folders and subfolders as necessary to store them; the system hour and date will be used when creating folders and can be used when naming the Son32 files. The user must be able to select the signals that he wants to store and pause/resume the storage.

- **Plotting graphs**

A real time plotting system must be used in order to display signals on screen in real time. Since plotting these graphs with a frequency of 20KHz can be slow when using too many signals (which implies too many points of measurement on the screen), a sub sampling method can be applied to solve this. The user must also be able to "rewind" the visualization without stop plotting the graphs; this means that a buffer system must be used to store these points.

- **Trigger function**

Very often, the scientists are interested in stimulating an organism using an electrical current in order to observe its responses due to the stimulation. In this context, a trigger function similar to those presented in oscilloscopes can be of great help when observing these responses if the stimulation signal is synchronized and used as the trigger. The trigger function consists of a window for plotting graphs with a fixed duration. This view is reset and re-plotted every time the trigger activation criteria are met. Therefore, the user informs the signal that will activate the trigger, the corresponding level for activating it (for example, when a value greater than 1V is detected), how much time is shown before the trigger activation and how much time is

shown after the trigger activation. When creating a signal, the user must also be able to choose whether he wants to display it on the trigger window or not.

4.3 Specification

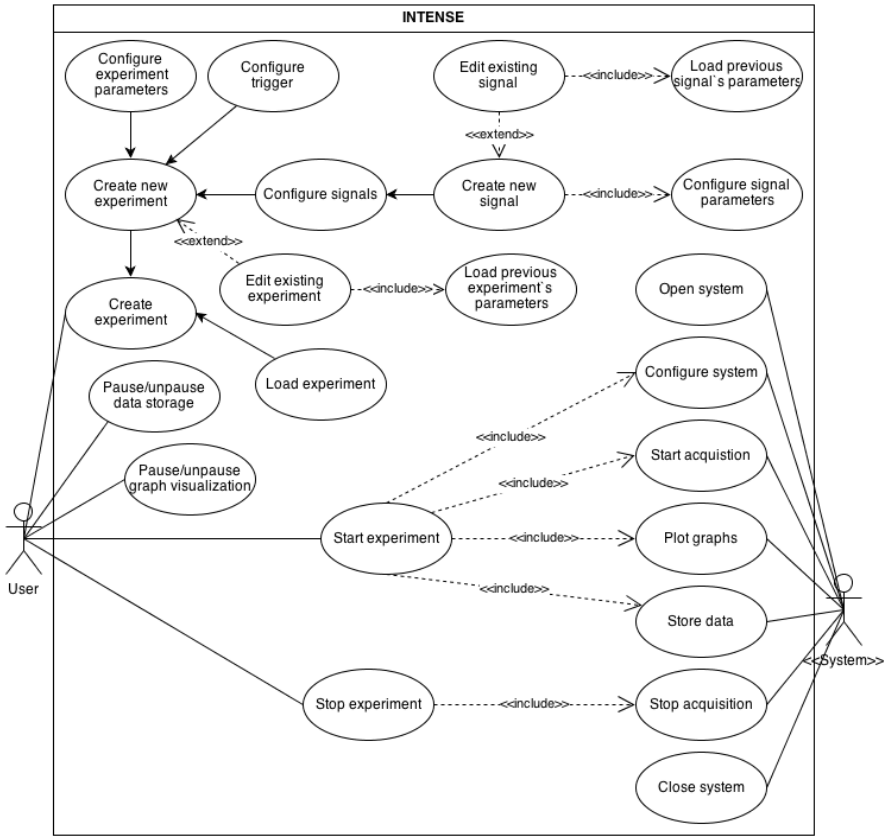

Figure 4.3 - Use case diagram

The use case diagram presented above was made after analyzing all the requirements. This diagram went through several modifications during the specification part of the project and contains all main functionalities that the software needs to present. The detailed description of each use case was omitted. Here only the main functionalities are shown.

5 CONCEPTION

In this chapter the main solutions and approaches used during the software development are presented. This also includes: the work environment; how to retrieve data pages; the characteristics of the main classes, their hierarchy and how they are organized; the file format used to save experiments; key concepts during the software development.

5.1 Software development environment

A SDE created at CEA was proposed in order to develop the software here presented. The main benefit that one may get from using such approach is the standardization of all library and framework versions for all of its users, which avoids compilation problems from machine to machine. Also, since other employees at CEA use this same SDE, a lot of algorithms and entire components are already implemented and ready to be used. A TortoiseSVN keeps the versioning and every file up to date. The list showed below presents the main tools that are available in the SDE.

	Tool	cmake 2.8.8	boost 1.5.1	cppunit 1.12.1	fieldtrip 3231	qt 4.7.4	qwt 6.0.1	son 7	tcl 8.6.0	tinyxml	Plxapi 6.2
	Keyword		BOOST	CPPUNIT	FT_BUFFER	QT	QWT	SO N	TCL	XML	PLX
WIN32	MSVS 2008										
	MSVS 2010										
WIN64											
LIN32	G++ 4.6.1										

Table 5.1 - SDE available features

The API named BIOMEART_SYS that was briefly presented before is an older implementation and was not part of the SDE. So a new library was created based on this API, inserted in the SDE and renamed as intensesys. Since the beginning, the idea was to create a library named intenseacq that would contain all classes that doesn't belong to the GUI and are intended to be reused (classes to model an experiment, signals and access the NeuroPXI system) and a top-level program named intensegui that would contain all GUI classes and bind every other library necessary.

5.2 Retrieving data pages

The library intensesys is a low level implementation. Through some of this library's functions it's possible to access and modify the configurations of the

NeuroPXI system and control the acquisition of signals. It presents one function that allows the developer to indicate a function or static method as the callback procedure that will be invoked every time that a data page is ready to be retrieved via DMA. In order to do that, a pointer to a function as the one showed below must be indicated as a parameter.

```
int notificationProc(void* data, int size, int type); /* or */
static int MyClass::notificationProc(void* data, int size, int type);
```

Every time this function is called the parameter type will inform which kind of data it contains (raw data ready to plot, error, warning or end of the acquisition notification). All data samples are received as a pointer to void, which needs then to be casted into a suitable type.

```
int BIOMEART_SYS_SetNotificationProcedure(
              int (*notificationProc)(void* data, int size, int type),
              char* error);
```

5.3 Class diagram

Here is a simplified class diagram of the solution proposed. Notice that the functions used to access the NeuroPXI system are C functions and not methods; however, this group of functions was represented here as a class named “BIOMEART_SYS C Functions” in the intensesys library because the goal of the class BiomeartHandler is to encapsulate all these functions. Many other classes and dependencies were omitted (as well as some third-party libraries, Boost, SON, some Qt dependencies and TinyXML).

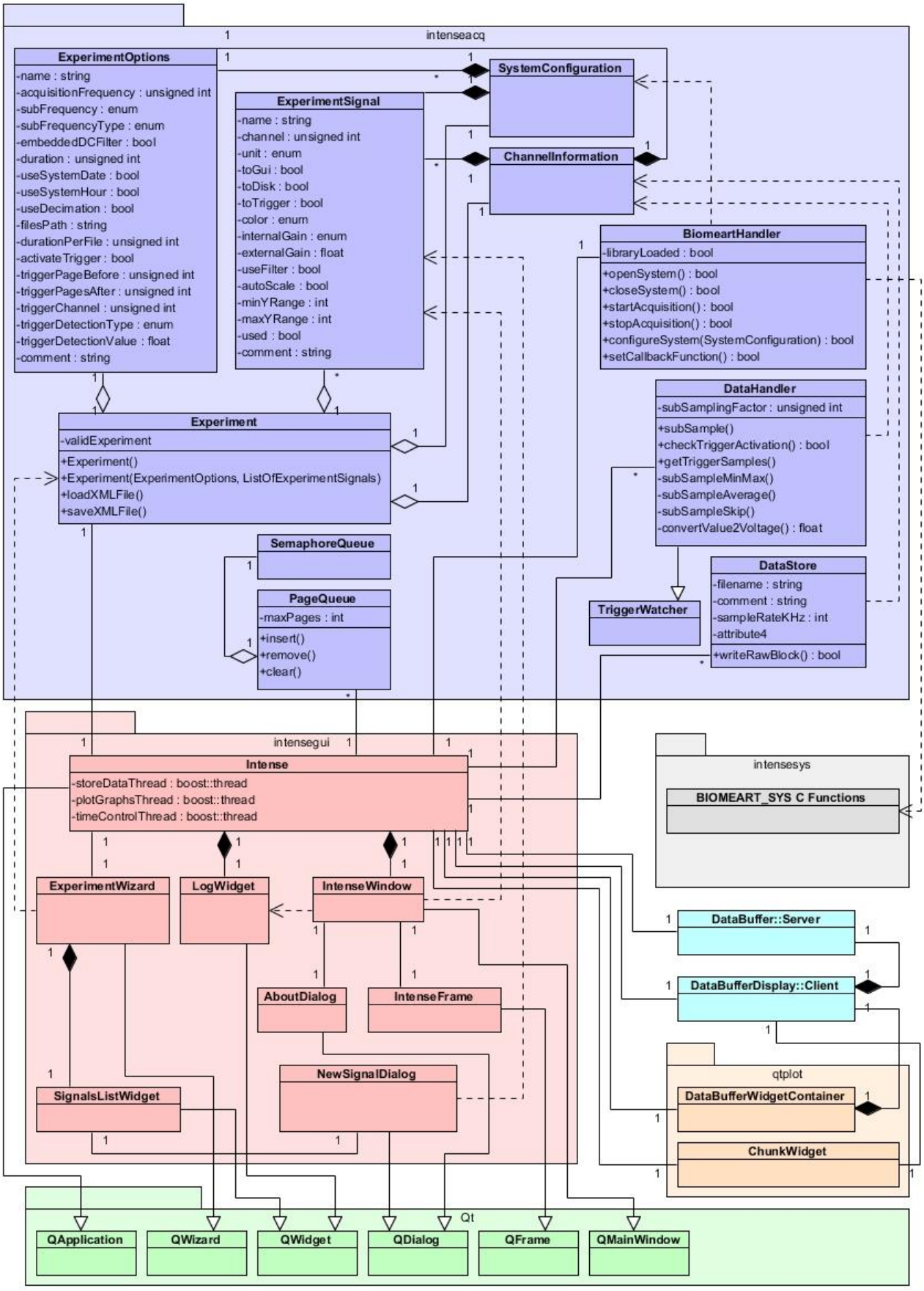

Figure 5.1 – Simplified class diagram

5.4 Characteristics of the main classes

Experiment: This class will contain every parameter of an experiment (ExperimentOptions and a list of ExperimentSignals). Its parameters can be initialized by either using its constructor, or loading a XML file. The attribute validExperiment will inform whether all parameters are valid or not after loading a XML file.

DataHandler: this class will be responsive for organizing data contained in a page into the format that the graph plotter will use. It will also perform the sub sampling when necessary and the value conversion (following the conversion table shown before) to display the values using their corresponding units (uV, mV or V). To make this conversion properly, it is necessary to take into account the signal's unit, the internal gain (gain configured by the user and provided by the AGNES) and the external gain (gain externally branched to the signal, provided by MCS uPA32m PGA 64 or any other amplifier). Since it is desired to see the value at the entrance of the system, we must divide the values received in the data page by the internal and external gains. Then, it is necessary to multiply this value by its unit (10^6 for μV, 10^3 for mV and 10^0 for V) in order to display the correct final value. The TriggerWatcher will contain information concerning the trigger activation; every time a new page is received we test the trigger activation criteria within the page to know whether we must activate the trigger or not (if the user chooses to use the trigger function).

DataStore: this class will be responsive for storing all data. Every time a new data page is received the proper method must be called to store all data in the Son32 file. The time stamp contained within each page will help informing where to record this page in the file. The possibility of splitting the experiment in several files must be present; to do this, all we have to do is create a new DataStore object and re-start storing data from where we stopped (using the page's time stamp).

PageQueue: a multithread queue implementation using semaphores for handling data pages.

BiomeartHandler: as mentioned before, this class will encapsulate all information concerning the BIOMEART_SYS C functions necessary to access the NeuroPXI system. To configure the system, this class uses a SystemConfiguration object that is generated based on an ExperimentOptions and a list of ExperimentSignal objects.

Chunck Widget: class responsible for plotting graphs on the screen. It is an assembly of QwtPlots instantiations along with a set of commands for controlling zoom, scale and other display properties. It also allows the display to be paused/rewinded to analysis without stopping the acquisition. This is done through a binary file that contains two pointers. One pointer is responsible for writing new incoming data into the file and the other for reading data previous written.

Intense: this is the main application that binds every component necessary, it will communicate with the user interface classes and start all main necessary threads.

5.5 The file format

To save every experiment after its creation, a XML file is used. The format of this file was defined in the beginning of the conception, containing every parameter regarding one ExperimentOptions and a list with every parameter regarding one or more ExperimentSignal. The parameters of an Experiment object can then be initialized using the read values. Here is an example of the format used by Intense to save/load an experiment.

```
<?xml version="1.0" ?>
<experiment>
    <options name="MANIP_VER">
        <acquisitionfrequency>20000</acquisitionfrequency>
        <subfrequency>5000</subfrequency>
        <subfrequencytype>average</subfrequencytype>
        <embeddeddcfilter>yes</embeddeddcfilter>
        <duration>300</duration>
        <usesystemdate>yes</usesystemdate>
        <usesystemhour>yes</usesystemhour>
        <usedecimation>yes</usedecimation>
        <filespath>X:/</filespath>
        <durationperfile>0</durationperfile>
        <activatetrigger>no</activatetrigger>
        <triggerpagesbefore>10</triggerpagesbefore>
        <triggerpagesafter>20</triggerpagesafter>
        <triggerchannel>23</triggerchannel>

<triggerdetectiontype>greaterthan</triggerdetectiontype>
        <triggerdetectionvalue>1</triggerdetectionvalue>
        <comment>Performed at CEA LETI 01/08/2013</comment>
    </options>
    <signalslist>
        <signal name="ENG1">
            <channel>192</channel>
            <unit>uV</unit>
```

```
            <togui>yes</togui>
            <todisk>yes</todisk>
            <color>green</color>
            <internalgain>1X</internalgain>
            <externalgain>10000</externalgain>
            <totrigger>no</totrigger>
            <usefilter>no</usefilter>
            <autoscale>yes</autoscale>
            <minyrange>0</minyrange>
            <maxyrange>2.5</maxyrange>
            <comment>Electroneurogram number 1</comment>
            <used>yes</used>
        </signal>
        <signal name="iStim">
            <channel>27</channel>
            <unit>uV</unit>
            <togui>yes</togui>
            <todisk>yes</todisk>
            <color>blue</color>
            <internalgain>10X</internalgain>
            <externalgain>0.125</externalgain>
            <totrigger>no</totrigger>
            <usefilter>no</usefilter>
            <autoscale>yes</autoscale>
            <minyrange>-5</minyrange>
            <maxyrange>5</maxyrange>
            <comment>No comment for this signal</comment>
            <used>yes</used>
        </signal>
        <signal name="MAN">
            ...
        </signal>
        ...
    </signalslist>
</experiment>
```

If the user chooses to edit an experiment all data contained in the XML file is read in order to create the respective classes instances. All information contained in the file is checked and if there is any error in the file format the user is informed that the file could not be open and the reason for so.

5.6 Used approach

Since the data flow is fast (one data page is received every 25.55ms) it is necessary to avoid data loss. So, after the callback function defined is invoked, it must be available again as soon as possible. In conclusion, this function cannot be too

complex. The strategy is to store all data received in a FIFO queue inside the callback function since this process is fast. Then, two threads are created using Boost: one for storing data on disk and another one for plotting graphs in real time. The idea is to give priority to the storage thread, so two queue objects are used. The storage thread is the consumer of the main queue; after this, in this same thread, the same data page received is stored in the second queue, which will be used to plot the graphs. Notice that semaphores must be used to synchronize the queue (the problem known as single producer/single consumer is presented in this case).

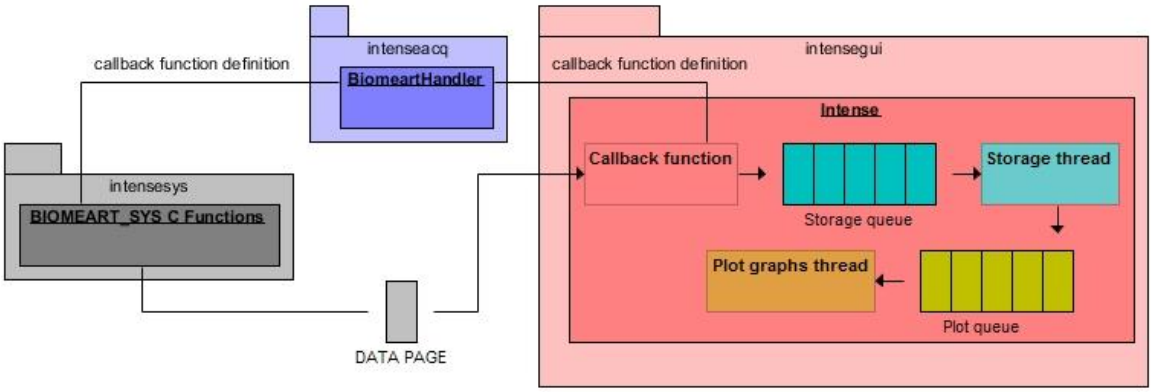

Figure 5.2 - Illustration of the queue strategy used for storing data pages

There is an important detail that one might ask: if the consumer is slower than the producer and since experiments can last of hours, then, at some time, the buffer stored using the queue will certainly overflow. But this is not why the queue implementation is used. The problem that the queue solves is that the operating system might make a context switch at any moment. When the software loses the context, the queue is used to absorb the incoming pages while waiting to regain its context. The methods themselves are fast enough to avoid a buffer overflow.

Every page that is received contains a fixed size. This means that every channel data will be received whether the channel is activated or not (it might contain some useless data). The pages contain a header, the data samples part, and the end of page indicator. Among the information that the header will contain is the time stamp and the page number; this is important to keep track of the pages that are being received and to be sure that there was no data loss.

5.7 Data storage

Son32 is a complete library for storing data using the Son32 file format. The software Spike2, is used for reading this type of file. Since this library provides every function necessary, storing data becomes easier. A DataStore object instantiated in the storage thread creates a new file and concatenates data to it as new pages are received. In the constructor it is indicated the frequency used, all channels are created and the necessary buffer space is allocated. Then, every time a new page is received, the samples contained within the page are re-organized in order to retrieve only the values of the channels that are being used and concatenated to the file with the function SONWriteADCBlock for each channel. All necessary functions of the Son library are encapsulated in the DataStore class. A fast access hard disk that spins at 15,000 RPM is generally used for the experiments, since the callback function can be blocked if we use a slow hardware.

As mentioned before, before storing the data received in the Son32 file, the samples must be re-organized, since the page format and the format used by SONWriteADCBlock function are different. When storing these samples, we must organize the array in such a way that all samples of the first channel are placed in the first positions of the array. The group of samples of the next channel will be placed in the following positions of the array and so on.

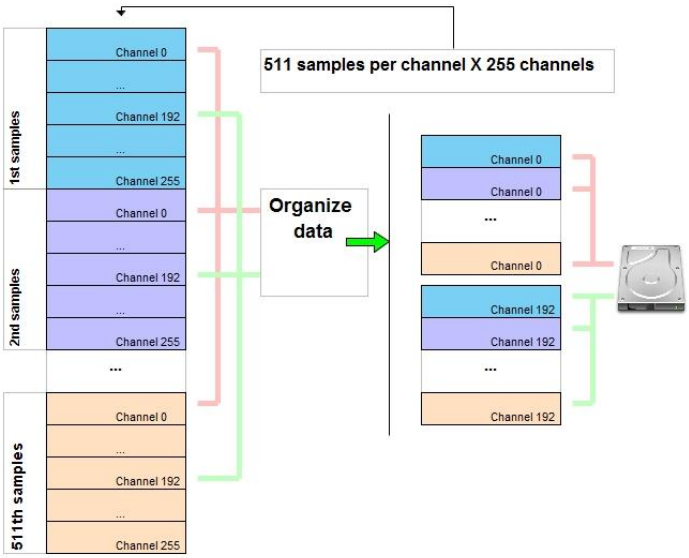

Figure 5.3 - Samples from a data page being organized for an experiment with two signals connected to channels number 0 and 192 to be stored on disk

When plotting graphs, the samples must be organized in the same way as the data page format, in which we present the first sample of all channels in the first positions of the array; then the second sample of all channels in the following positions of the array and so on. This method will be explained later.

5.8 Plotting graphs

The Qwt library (provided by Qt) was used for plotting graphs. The class named Qtplot::ChunkWidget provides a plotting system that works by declaring several QwtPlots and binding them together in a single view, displaying also the controls for zooming in and out, hiding channels, modifying scales and adding/deleting markers, as well as a buffer system that allows rewinding the visualization.

However, since plotting graphs smoothly during each interruption is hard due to the high frequency that is used and blocking the main application is not an option, a separated process is created and embedded into the main one. In order to do this, it is declared a DataBuffer::Server object and a DataBufferDisplay::Client object, passing the same channels' information as parameter for both of them.

These two objects are bound together, since it is informed the server's name to the client object as well. The client object will then create a XML file to save all channels' information in a temporary folder, and create a new process named databufferdisplayclient.exe. This process will contain a Qtplot::ChunkWidget object for plotting graphs. However, the main application won't communicate with the new process created (no commands can be sent via GUI); the only data that this process can receive is data chunks to be plotted using a shared memory zone. Every time that new data to plot is received, it is copied to the shared memory by using the server's method called SendChunk. The process previously created will then be able to retrieve and plot this data. This avoids the main application from being blocked. To embed the databufferdisplayclient.exe process in the main window, it is used a Qtplot::DataBufferWidgetContainer object, informing the client's pointer as a parameter.

The data page format is the same as the one used by Qtplot::ChunkWidget for plotting graphs. However, all data page received contains samples of every channel. So we need to iterate through every page and retrieve only the samples of the channels that

are activated in the GUI. The image below shows an example of a simple experience that uses only two signals.

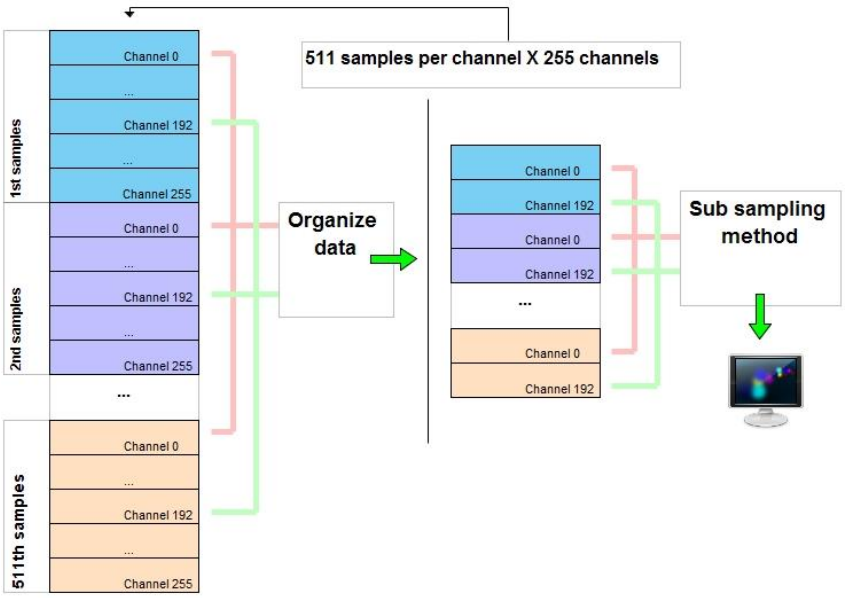

Figure 5.4 - Samples from a data page being organized for an experiment with two signals connected to channels number 0 and 192 to be displayed on GUI

5.9 Display improvements and sub sampling methods

The components used for plotting graphs offer a great precision and, theoretically, an unlimited number of channels can be created. However, when plotting these graphs using a frequency of 20KHz, the display can suffer from slowdowns due to the huge number of points of measurement sent each time (511 points per 25.55ms for each channel). To avoid this problem, two solutions were created.

The first one is named decimation and it calculates how many points of measurements will be shown based on the screen's size. If the user has a tiny screen, fewer points are shown. Nonetheless, this doesn't affect the buffer system, which means that every point of measurement is buffered; so every point is loaded and displayed when the user rewinds the visualization. This function can be activated or not by the user using the parameter usedecimation.

The second one is used when the user is more interested in observing the wave form only, absent need of checking every single point with maximum precision. In this case, to make the signal display more fluid, three sub sampling algorithms were implemented in the DataHandler class. The user must inform the sub sampling frequency desired that can be either 20KHz (no sub sampling used, every point of

measurement is displayed), 10KHz, 5KHz, 4KHz, 2KHz, 1KHz or 500Hz. Then, based on the user's choice, the sub sampling factor is calculated. The user can also choose between three different sub sampling methods. The most important of these three algorithms is the one called "average", in which a number of samples is gathered and only their arithmetic mean is displayed. The figure shown below demonstrates these three sub sampling methods when using a sub sampling frequency of 4KHz.

$$sub\ sampling\ factor = 20KHz/4KHz\ = 5\ (n\ samples\ shown\ below)$$

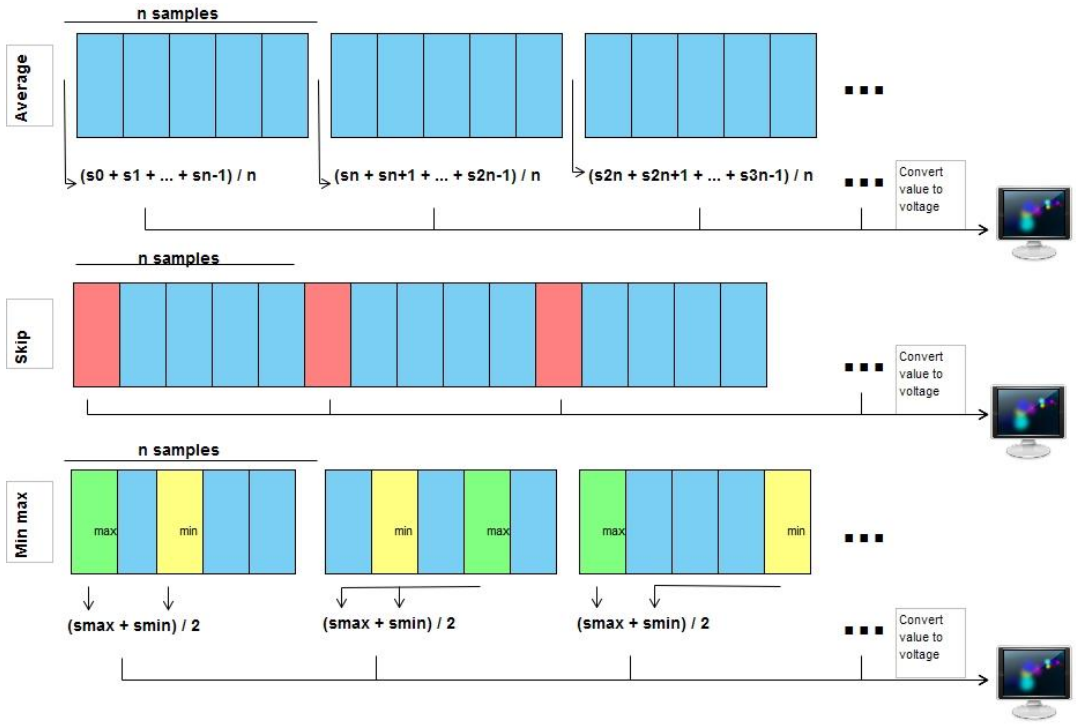

Figure 5.5 - Sub sampling methods for a sub sampling frequency of 4 KHz

5.10 Value conversion

Before being displayed, every point of measurement must be converted to its corresponding value in volts according to the table shown before (bit conversion table). This conversion is performed just before displaying these points, as shown in the figure above (the box "Convert value to voltage"). A simple yet powerful algorithm that uses the same type of conversion that the Son32 library proposes was developed in order to do this. It's also at this moment that the internal gain, external gain and unit must be taken into account.

```
/* 2 is the scale format correction. 65536 is the maximum range for a
short variable. 10/65536 is the same conversion used by Son32*/

const float DataHandler::bitToVoltFactor = 2.0F*(10.0F/65536.0F);

float DataHandler::convertValueToVoltage(short valueToConvert, int channel)
{
    float gainFactor = ( m_chanInfo.unitMultiplier[channel] /
                        (m_chanInfo.internalGain[channel] *
                         m_chanInfo.externalGain[channel]) );
    return (valueToConvert*gainFactor*bitToVoltFactor);
}
```

For example, if we receive the first value of the bit conversion table, without any gain (internal and external gain equals one) and using the Volt unit:

	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2.5V	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

$$valueToConvert = 0001111111111111b = 8191$$

$$gainFactor = \frac{unitMultiplier}{(internalGain * externalGain)} = 1$$

$$\text{bitToVoltFactor} = 2 * \left(\frac{10}{65536}\right) = 0.00030517578125$$

$$returnValue = valueToConvert * gainFactor * \text{bitToVoltFactor} =$$

$$= 8191 * 1 * 0.00030517578125 = \mathbf{2,49969482421875}$$

Here is a glance at the code responsible for organizing data, sub sampling it and converting the values. This is the algorithm that calculates the average when sub sampling.

5.11 Testing signal display

After finishing all these steps, the chain of functions presented and the widget used to display all the signals could finally be tested. In order to perform all tests, it was used several signals with well-known frequencies and ranges with a function generator and a signal generator named ME/W-SG from MultiChannel Systems. This generator can simulate spikes, sinus waves, atrium ECG, ventricular ECG, as well as other wave forms.

Figure 5.6 - ME/W-SG signal generator from MultiChannel Systems

Different gains configured and applied by the AGNES (internal gain) and the gain provided by external amplifiers (external gain) were also applied to be sure that the correct values that were injected at the entrance of the system were being observed. A chronometer was used in multiple tests to check if the time visualization was really accurate and the markers provided by the widget were used to see if the frequencies of the signals were correct. After being sure that the signal display worked properly, some other small improvements were made such as adding different colors (chosen by user) and allowing the user to register the scales in the file, avoiding the need of reconfiguring them each time the software is launched.

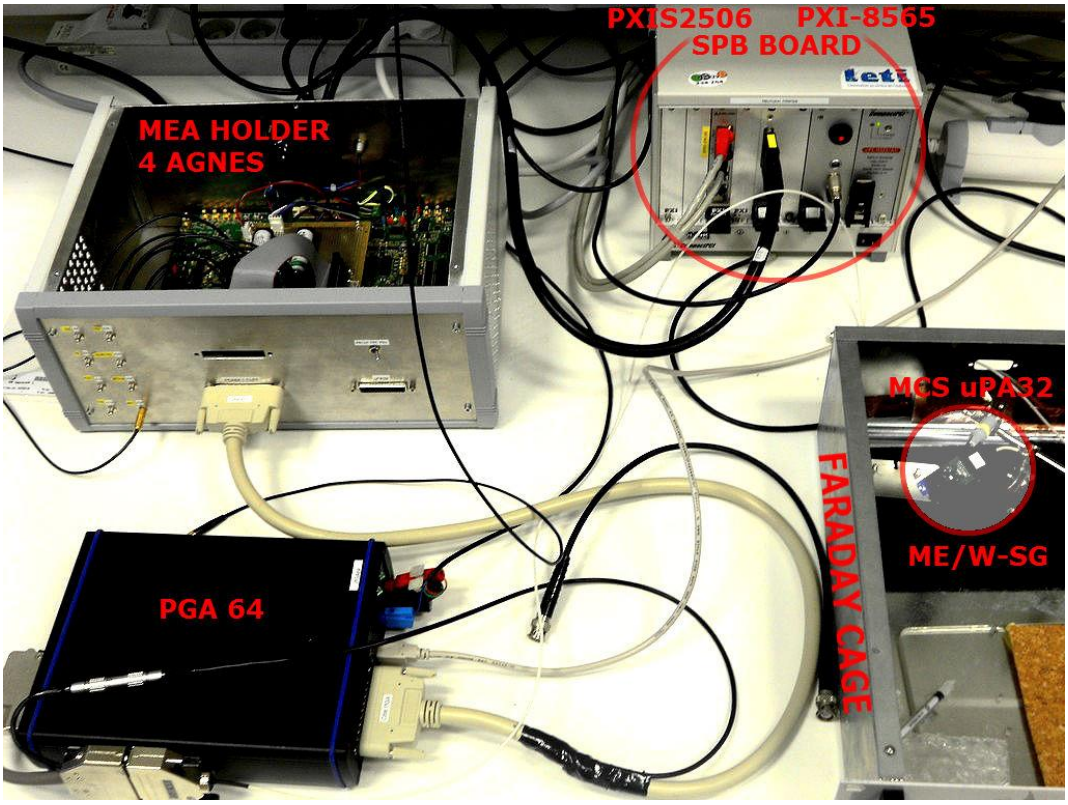

Figure 5.7 - Acquisition chain used for testing the Intense GUI

In the tests shown below some hippocampal neuron spikes were simulated with the ME/W-SG (shown in green) and a manual button with a battery was also usedto generate some pulses (shown in red).

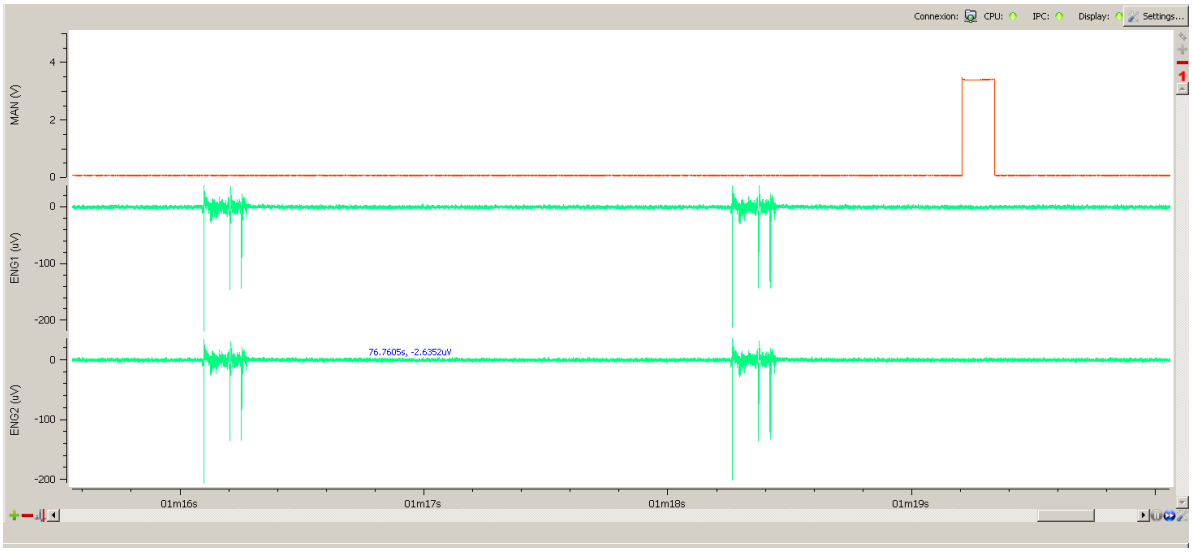

Figure 5.8 – Test of signal display with ME/W-SG and a button using Intense

5.12 Trigger function

The trigger function is a simpler implementation, since it uses only a Qtplot::ChunkWidget object for plotting graphs because is not activated every 25.55ms like the graph display and it usually presents less channels. Every time a new page is received, a DataHandler's method will verify the trigger activation criteria. If the trigger should be activated, the trigger display is reset and it re-plots the window with n1 pages received before the trigger activation and the following n2 pages after the trigger activation (informed by the user).

In order to do this, is necessary to keep the n1 pages that are received in a buffer. In this case, the best solution is to use a ring buffer (also called circular buffer) with a size of n1. If this buffer is full, the first page that was allocated will be replaced with the new one received. Once the trigger is activated, all this pages are sent to the trigger display and the buffer is empty. The following n2 pages are then sent to the trigger display normally as they are received. The trigger function will always use a plot frequency of 20 KHz, which means that every single point of measurement received is used.

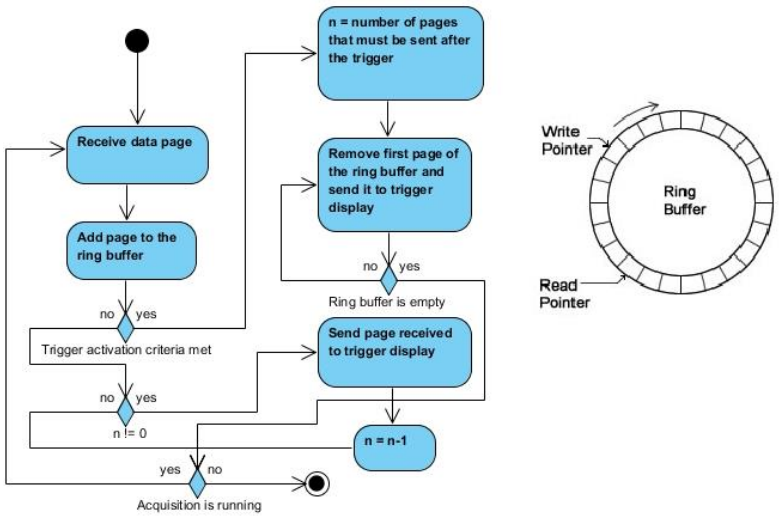

Figure 5.9 - Activity diagram for trigger function and an illustration of a ring buffer used to store data pages

5.13 Main window

The main window can be divided in:

Menu bar: located at the uppermost part of the window, it contains all possible commands that the user can send to the main application. The commands are available according to the program's flow. For example, at the beginning of the execution, the "Edit experiment", "Start acquisition" and "Stop acquisition" are blocked. After an experiment is created or loaded it's possible to edit it and start the acquisition. When the user starts the acquisition these commands previously mentioned are blocked and the command "Stop acquisition" becomes available, and so on. This was made in order to help the user using the software, since only valid commands will be accessible.

Toolbar: located just beneath the menu bar, it contains the shortcuts for the most important commands of the menu bar. These shortcuts are: "Create new experiment", "Load experiment", "Edit current experiment", "Pause/resume storage" (applied for all signals, activating/deactivating the method in the DataStore object that stores data), "Start acquisition" and "Stop acquisition".

Signal tree view: Located at the left part of the window, contains information of all signals being used in the current experiment.

Mdi area (multiple document interface): Contains the embedded process responsible for plotting graphs and the trigger display.

Log widget: located at the bottom of the window, it consists of widget with three tabs. The first one is a log that contains information about every operation performed by

the software. It contains four different levels of message: “success” in green, “information” in blue, “warning” in orange and “error” in red. The second tab generates a list view of every Son32 file generated, with the hour and the status. The user can open the respective folder by double clicking the list item. The third tab is also a list view, but it contains information about possible data losses. If for some reason the callback function couldn’t be invoked at a desired time and it couldn’t retrieve the data page, this tab informs the hour that this happened, the expected page that was to be received and the page that was actually received last time.

Status bar: located at the bottommost part of the window, the status bar informs the current Son32 file that is being written (path and file name). It also has a LED that blinks every 0.5s to inform if data is being stored (Son32 file being written), a timer that informs how much time is left until the end of the experiment and a progress bar.

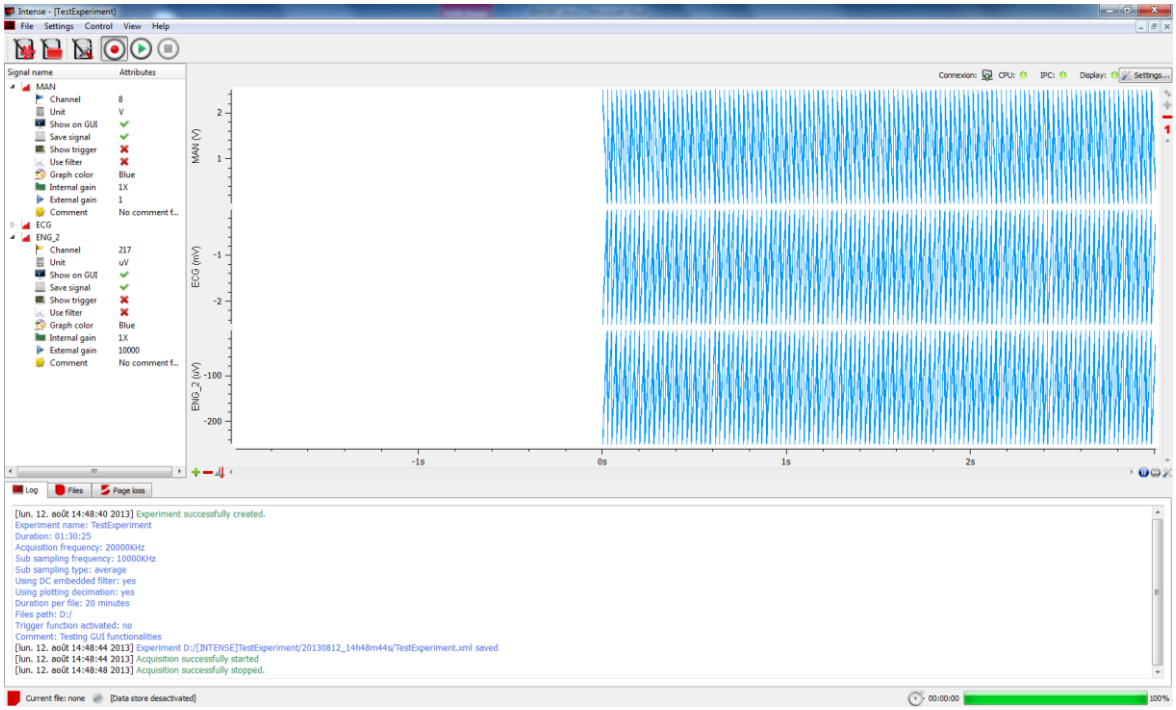

Figure 5.10- Main window of Intense

5.14 Experiment wizard

To make it easier for the user to create a new experiment, a wizard assistant was developed using Qt; which is basically a step-by-step guide.

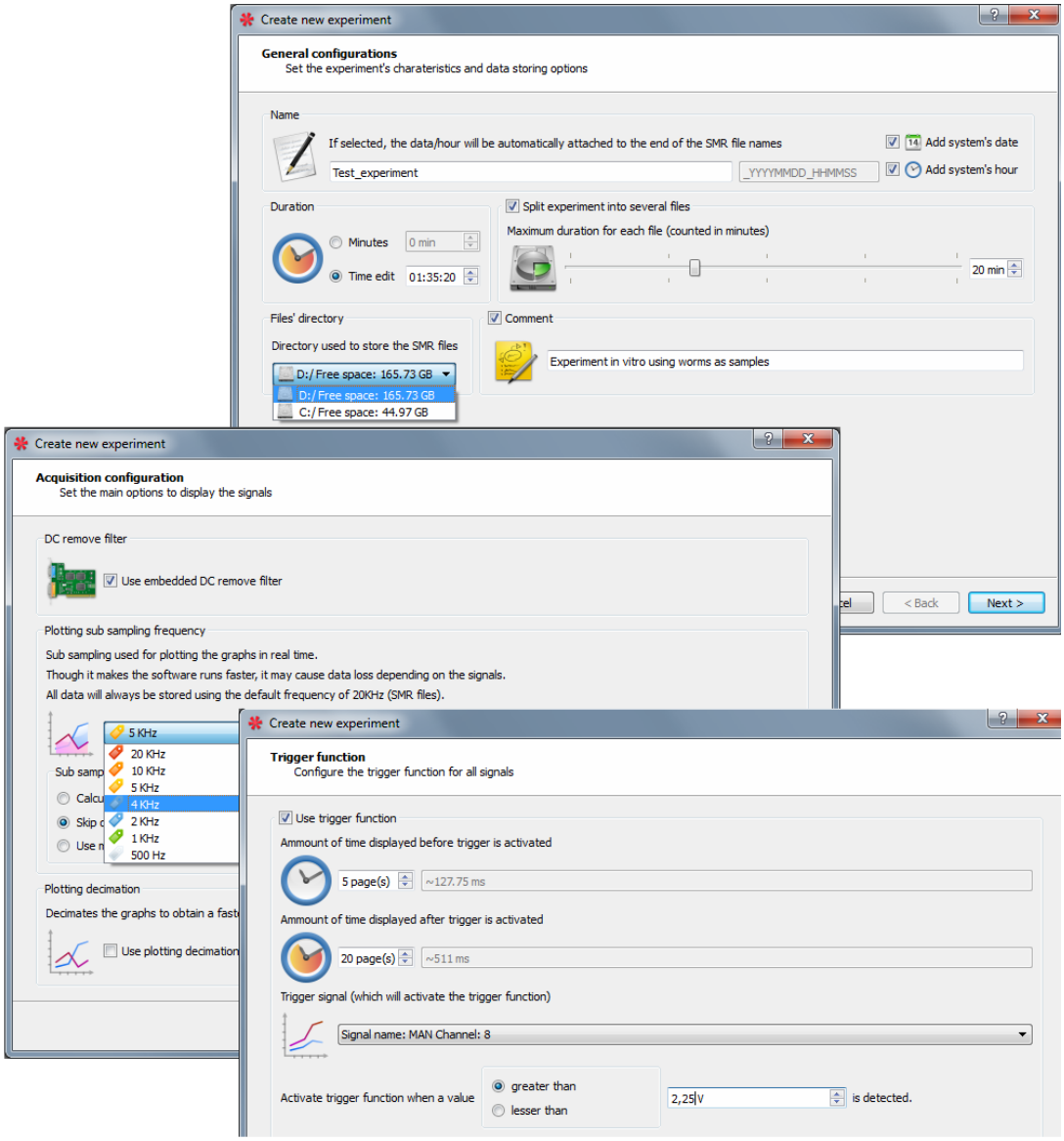

Figure 5.11 – Three of the four pages that the wizard contains, displaying general configurations, sub sampling methods, decimation option, the DC filter (performed by the FPGA) and the trigger configuration

5.15 Managing signals

One of the most important aspects of this project was the signal creation. The goal here was to make a complete configurable signal creation method and manageable system. A SignalsListWidget object is responsible for managing the signals, allowing the user to create new signals, edit existing signals, delete signals, as well as load a list of signals from an XML file of an older experiment. To create/edit signals a NewSignalDialog object is used, which allows the user to configure every parameter that a signal contains.

For all components, every kind of verification is performed. For example, in the NewSignalDialog only the available channels are shown. The SignalsListWidget is embedded in the third experiment wizard's page. The user can create up to 256 signals, but he doesn't need to use all of them, which means that only the signals in the list of used signals will be taken into account for the experiment.

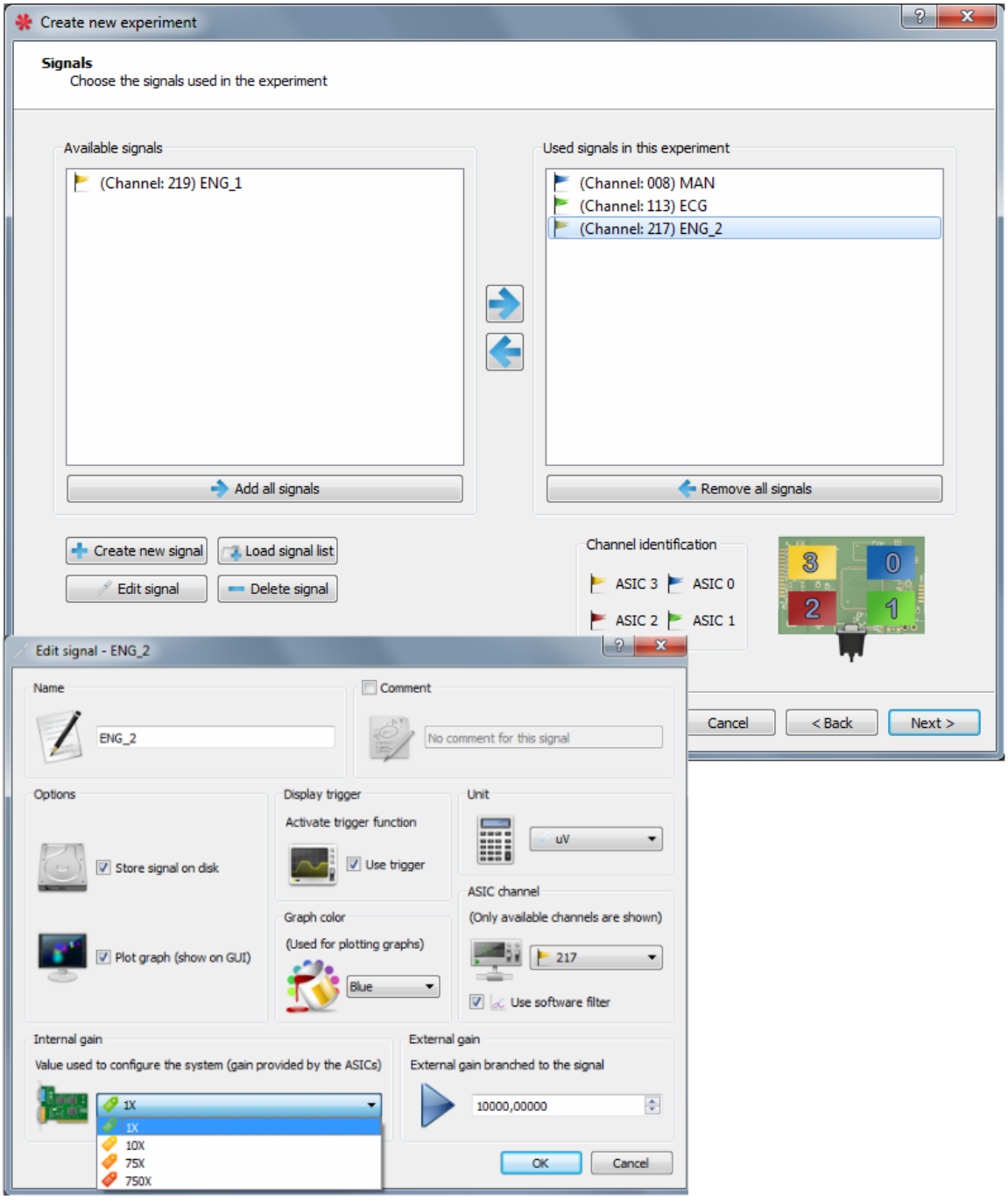

Figure 5.12- Signal manager and the edition of an existing signal

5.16 Verification and validation

Throughout the whole process of conceiving the software, discussions took part with everyone involved in the project that would use this software and knew exactly what functionalities it should have (from a user point of view). The engineers involved in the project helped giving a proper guidance for developing software that attended to their needs. Weekly reunions were scheduled in order to keep track of the work and also to refresh the focus of the main goals, keeping them up-to-date.

During the whole work the goal was to create an extensible code, trying to abstract the maximum possible from all classes that I created (mainly **intenseacq**'s classes). This way, if someone needs to extend and modify the code, this won't present any problem. This goal was achieved by creating a reusable code mainly due to the conception of separate libraries and a program dedicated only for the GUI that could be easily tested separately. This means that **intenseacq** is totally independent from the GUI and it contains everything to model an experiment, its signals and access the NeuroPXI system. , It will be easier if anyone wants to modify or create a new GUI by using the **intenseacq** library since the program **intensegui** only contains Qt classes that give the GUI its format with its windows, dialogs and wizard that will construct **intenseacq**'s objects and use them.

Most part of the approach was based on unit testing, which means that every single component was tested as soon as it was ready. Another practice adopted was redoing every test as soon as more pieces of code and functionalities were added. Other engineers also checked the software constantly and pointed out the modifications that were desired.

6 STUDY CASE

When the software was practically finished, three engineers used it to provide feedback of what they thought. The experiment was performed using a worm as the test subject. A worm is a good in vivo model to test the cuff electrodes that register the ENG signals and the chain of acquisition using NeuroPXI with a living being. The worm is anaesthetized and placed in a Faraday's cage along with the preamplifier. The main goals of this experiment are to register unitary activities generated around the animal's body with the cuff electrodes and to register the spontaneous unitary activities in response.

The worm is completely clean and soaked in an ethanol 10% solution from five to ten minutes, making sure that is completely motionless and kept in a moist place. The anesthesia should last about thirty minutes. The amplifier is configured with a gain of 1,000X, resulting in a final gain of 10,000X (10X from the pre-amplifier) for the ENG signals (internal gain equals 1x, external gain equals 10,000X).

For this experiment the interest lied particularly in stimulating the subject and observing the responses generated. A simple diagram of the architecture conceived for generating the stimulation is shown below. It makes use of commercial equipment and some other devices developed at CEA. A trigger signal marks every time that the stimulation is applied; this signal is called SyncStim, and it has a gain of 0.125x (internal gain equals 1x and external gain equals 0.125x), since we use a voltage divider with a factor of 8 instead of an amplifier. The stimulation frequency equals 0.5Hz.

Figure 6.1 - Modular architecture for generating the stimulation

Several external signals are connected to the NeuroPXI system in order to verify that the acquisition chain works properly. The cuff electrode is then placed in phosphate-buffered saline solution and the RMS tension around the cuff is measured, as well as the RMS current in a resistance in series with the cuff.

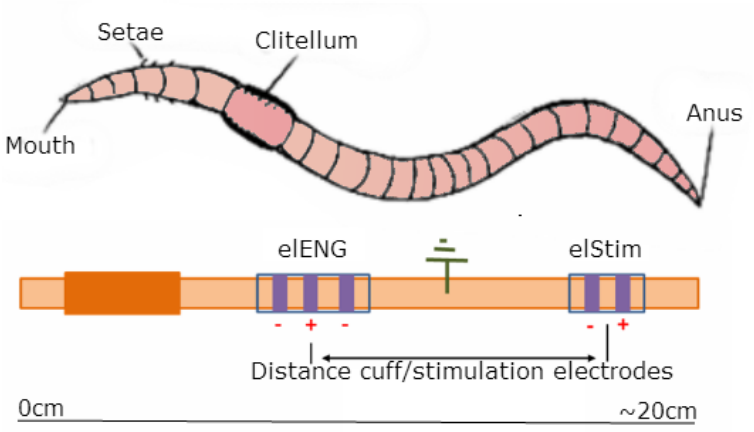

Figure 6.2 - Placement of the cuff and the stimulation electrodes on the worm

After all measurements and calibrations are performed by the engineers, the experiment is ready to begin. In this context, the engineers were interested in testing the trigger function present in the GUI. A new experiment was created using the software, all signals that we needed and configured the trigger window to display two ENG signals that we were interested in and the SyncStim signal, which was the signal chosen to activate the trigger. The trigger activation value was set to values greater than +1V.

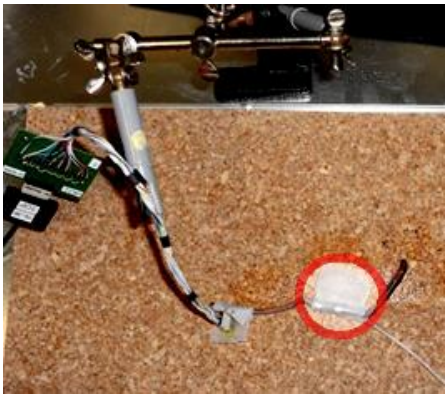

Figure 6.3 - Subject ready for the experiment. The red circle shows the stimulation cuff electrode placed around the worm. The other electrode in the left is responsible for registering the ENG signals

The trigger window is refreshed every time a high event that surpasses +1V is detected by observing the SyncStim signal. When using the trigger function, the notion of time is not important for the engineers; what they want to see is the responses after the stimulation. The whole experiment can be viewed using the Son32 files afterwards.

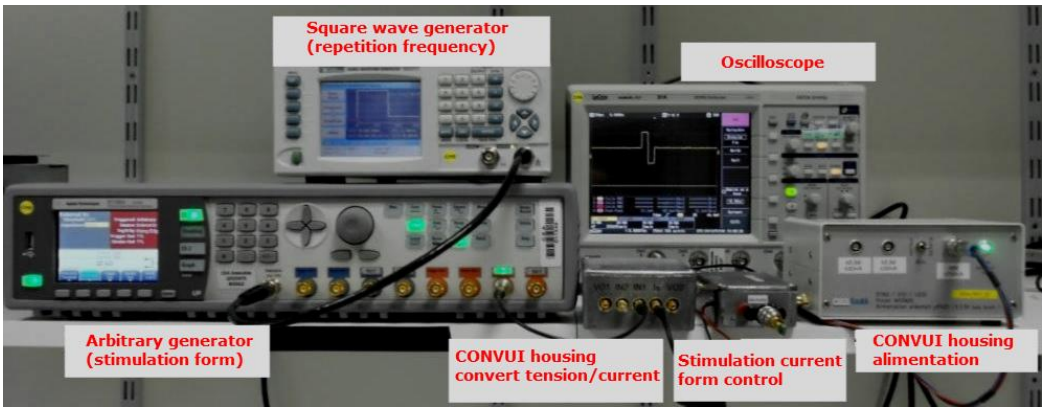

Figure 6.4 - Stimulation chain for the experiment with the worm

This experiment was successfully performed and they could clearly see the stimulation being applied to the animal and the responses in the ENG signals due to this stimulation. In this case, two ENG signals were used; no filter was used for the second signal, but a high pass 400Hz filter was applied to the first ENG to remove traces of ECG (the user can activate or not the filtering for each signal, the high pass filter is performed by the BIOMEART_SYS API, its coefficients were calculated using MATLAB and are located in a TXT file). The DC remove filter was not used (the DC filter is applied by the FPGA).

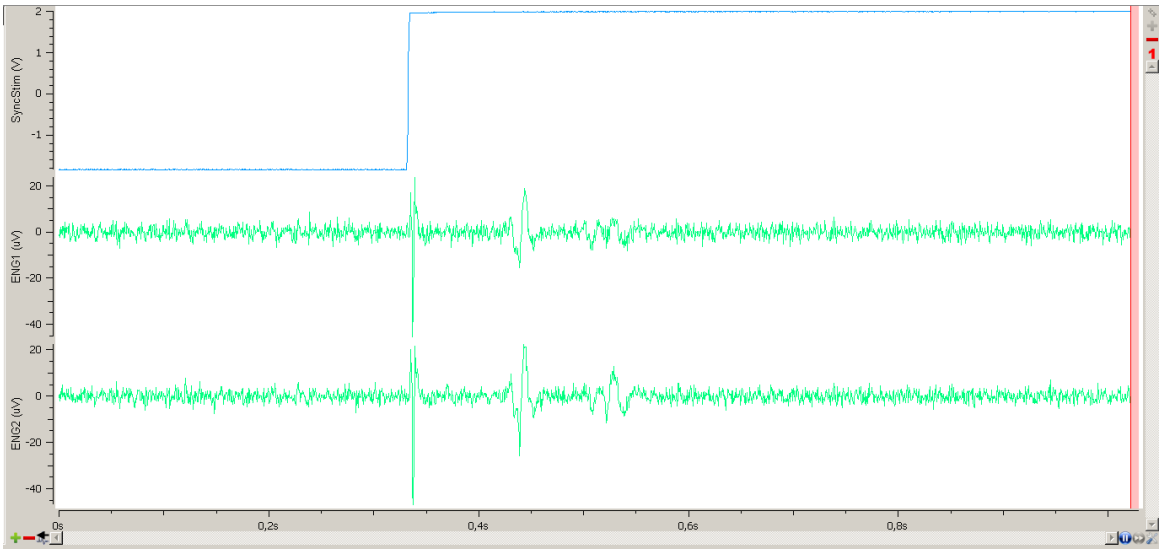

Figure 6.5 - Stimulation being applied and its action potentials for two ENG signals using Intense GUI's trigger function. The first oscillation is the stimulation artifact, followed by two action potentials

After seeing the responses they wanted, the engineers retrieved the Son32 files generated containing every data at 20KHz. These files are then submitted to posterior analysis using MATLAB and Spike2 with the goal of trying to understand the interaction between the stimulation and the action potentials generated.

The two action potentials that can be seen are due to the fact that the worm possesses two large fibers along its body that respond to the stimulation. This experiment is much simpler than the one with pigs, which offers a closer environment and results in comparison with a human organism. However, the test with worms as subjects allows a fast set up and easier achievable results for testing purposes.

7 CONCLUSION

The whole project in which the software here presented is inserted is interesting, taking into account that many fields of knowledge converge in order to achieve a common goal. Technologies nowadays encompass many different means of identifying the nature of conditions to allow interventions to increase the life span or the quality of life, either by the use of new technological devices, pharmacological methods or biological methods.

The experiment previously presented shows that the software is ready to be used and stable to perform the necessary tasks. The trigger function demonstrated itself useful when looking for particular events among the signals. Even if this is a long-term project that still has some implementation parts to be finished, the chances of the whole hardware and its software to become a commercial version of a medical device are favorable.

Surely, software like this one always provides room for improvements. Many sketches were planned to be included in the GUI; however, they had to be postponed due to the little time available for all implementation. A visual display of the cuff electrodes that would communicate with its corresponding channel was desired. The possibility of providing a user interface for adding wave marks to the Son32 files is also absent. However, the current GUI provides every function previewed at the beginning of the project and a proper source code that is easy to modify and add new functionalities in the future.

Figure 7.1 - Intense GUI's logo

REFERENCES

[STROUSTROUP 2000] B. Stroustroup. **The C++ Programming Language: Special Edition**. 3rd edition**.** Addison-Wesley. 2000.

[BLANCHETTE 2008] J. Blanchette, M. Summerfield. **C++ GUI Programming with Qt 4**. 2nd edition. Prentice Hall. 2008.

[CAMBRIDGE 2012] **Spike2 for Windows Version 7** <available at http://www.ced.co.uk/img/Spike7.pdf>: last access: June 2013.

[MULTICHANNEL 2013] **Multichannel Systems ME/W-SG** <available at http://www.multichannelsystems.com/sites/multichannelsystems.com/files/documents/data_sheets/ME_W-SG.pdf>: last access: July 2013.

[MULTICHANNEL 2011] **Multichannel Systems PGA 64 Manual** <available at http://www.multichannelsystems.com/sites/multichannelsystems.com/files/documents/manuals/PGA_Manual.pdf>: last access: July 2013.

[MULTICHANNEL 2012] **Multichannel Systems uPA32 Datasheet** <available at http://www.multichannelsystems.com/sites/multichannelsystems.com/files/documents/data_sheets/%C2%B5PA32_Datasheet.pdf>: last acces: July 2013.

[BOOST 2013] **Boost 1.5.1** <available at http://www.boost.org/doc/libs/1_51_0/>: last access: August 2013.

[QT 2013] **Qt 4.7 Documentation** <available at http://qt-project.org/doc/qt-4.7/>: last access: August 2013.

[BIOMEA 2007] G. Charvet, O. Billoint, L. Rousseau, B. Yvert. **BioMEA™: A 256-channel MEA system with integrated electronics** <available at http://www.incia.u-bordeaux1.fr/IMG/pdf/2007charvetembsconf.pdf>: last access: December 2014

[BIOMEA 2010] G. Charvet, O. Billoint, S. Gharbi, M. Heuschkel, C. Georges, T. Kauffmann, A. Pellissier, B. Yvert, R. Guillemaud. **A modular 256-channel Micro Electrode Array platform for in vitro and in vivo neural stimulation and recording** <available at www.incia.u-bordeaux1.fr/IMG/pdf/2010charvetembsconf.pdf>: last access: December 2014

[NEUROPXI 2012] S. Bonnet, J.-F. Bêche, S. Gharbi, O. Abdoun, F. Bocquelet, S. Joucla, V. Agache, F. Sauter, P. Pham, F. Dupont, F. Matonti, L. Hoffart, S. Roux, M. Djilas, B. Kolomiets, R. Caplette, F. Chavane, S. Picaud, B. Yvert, R. Guillemaud. NeuroPXI: **A real-time multi-electrode array system for recording, processing and stimulation of neural networks and the control of high-resolution neural implants for rehabilitation** <available at: http://www.sciencedirect.com/science/article/pii/S1959031812000140#>: last access: December 2014