An Optogenetic Headstage for Optical Stimulation and Neural Recording in Life Science Applications

Mémoire

Reza Ameli

Maîtrise en génie électrique
Maître ès sciences (M.Sc.)

Québec, Canada

Résumé

L'optogénétique est une nouvelle méthode de contrôle de l'activité neuronale dans laquelle la lumière est employée pour activer ou arrêter certains neurones. Dans le cadre de ce travail, un dispositif permettant l'acquisition de signaux neuronaux et conduisant à une stimulation optogénétique de façon multicanale et temps-réel a été conçu. Cet outil est muni de deux canaux de stimulation optogénétique et de deux canaux de lecture des signaux neuronaux. La source de lumière est une DEL qui peut consommer jusqu'à 150 milliampères. Les signaux neuronaux acquis sont transmis à un ordinateur par une radio. Les dimensions sont d'environ 20×20×15 mm^3 et le poids est de moins de 7 grammes, rendant l'appareil utile pour les expériences sur les petits animaux libres. Selon nos connaissances actuelles, le résultat de ce projet constitue le premier appareil de recherche optogénétique sans-fil, compact offrant la capture de signaux cérébraux et la stimulation optique simultanée.

Abstract

Optogenetics is a new method for controlling the neural activity where light is used to activate or silence, with high spatial and temporal resolution, genetically light-sensitized neurons. In optogenetics, a light source such as a LED, targets light-sensitized neurons. In this work, a light-weight wireless animal optogenetic headstage has been designed that allows multi-channel simultaneous real-time optical stimulation and neural recording. This system has two optogenetic stimulation channels and two electrophysiological reading channels. The optogenetic stimulation channels benefit from high-power LEDs (sinking 150 milliamps) with flexible stimulation patterns and the recorded neural data is wirelessly sent to a computer. The dimensions of the headstage are almost 20×20×15 mm^3 and it weighs less than 7 grams. This headstage is suitable for tests on small freely-moving rodents. To the best of our knowledge, this is the first reported fully wireless headstage to offer simultaneous multichannel optical stimulation along with multichannel neural recording capability.

Table of Contents

List of Tables

List of Figures

List of Acronyms

A/D	Analog-to-Digital Converter
AC	Alternating Current
ADC	Analog-to-Digital Converter
AFE	Analog Front End
ASIC	Application-Specific Integrated Circuit
BCI	Brain-Computer Interface
BMI	Brain-Machine Interface
BNC	Bayonet Neill–Concelman
CMRR	Common-Mode Rejection Ratio
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DD	Discrete Derivatives
DMA	Direct Memory Access
DSP	Digital Signal Processing
DSP	Digital Signal Processor
DWT	Discrete Wavelet Transform
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FM	Frequency Modulation
FPGA	Field-Programmable Gate Array
FSK	Frequency-Shift Keying
GFSK	Gaussian Frequency-Shift Keying
GND	Ground
IO	Input Output
ISI	Inter-Spike Interval
ISM	Industrial, Scientific and Medical
IT	Integral Transform
LDO	Low-Dropout (regulator)
LED	Light-Emitting Diode
LFP	Local Field Potential
MCU	Microcontroller Unit
NA	Not Available
NDD	Normalized Density Derivatives
PCA	Principal Component Analysis
PCB	Printed Circuit Board
PMU	Power Management Unit
PSRR	Power Supply Rejection Ratio
PWM	Pulse-Width Modulation
QFN	Quad-Flat No-Leads (package)
RF	Radio Frequency

RFI	Radio Frequency Interference
RMS	Root Mean Square
ROC	Receiver Operating Characteristic (curve)
RTOS	Real-Time Operating System
RX	Reception
SMD	Surface-Mount Device
SPC	Superparamagnetic Clustering
SPI	Serial Peripheral Interface
TEO	Teager Energy Operator
TX	Transmission
VDD	Positive Supply Voltage
VLSI	Very-Large-Scale Integration

List of Symbols

PC_i	i^{th} principal component
c_i	Weight of a certain vector on the ith principal component of the dataset
$s(n)$	Action potential waveform/vector
$DWT(u,j)$	$(u,j)^{th}$ coefficient of the discrete wavelet transform
$\Psi(.)$	Mother wavelet function
$DD_d(n)$	Discrete derivatives of order d of signal $x(n)$
IT_P	Average of the nonnegative phase of an action potential signal
IT_N	Average of the negative phase of an action potential signal
N_P	Number of nonnegative samples in an action potential waveform
N_N	Number of negative samples in an action potential waveform
V_{Ref}	Reference (mid-supply) voltage for instrumentation amplifiers and op-amps
μV(RMS)	Root mean square value of an electrical signal expressed in microvolts

Acknowledgement

I would like to sincerely thank my supervisors, Dr. Benoit Gosselin and Dr. Paul Fortier, at Laval University for their endless support during the past two years. Dr. Gosselin gave me freedom in my work and research. He allowed me to test and realize my ideas without any obstacles; he provided me with all resources he had and let me build my own experience. He is always bringing new ideas and challenges to his research group, that's how he keeps his lab such a lively place to study, to research and to grow. Dr. Fortier was always there to help me and had time for me to have pleasant scientific discussions with him. Every single time I met him, I learnt a new way of thinking about mathematics. I would like to thank him for all his support. I learnt a lot from him… I would also like to express my gratitude to Dr. Amine Miled who accepted to be part of the jury that evaluates my thesis.

During my studies at Laval University, I had the chance to work with many great colleagues without whom this thesis would have never been as it is today. I would like to specially thank Gabriel Gagnon-Turcotte and Alireza Avakh Kisomi, my friends and colleagues that directly helped me with my M.Sc. project. I also would like to thank Hadi Bahrami and Abdollah Mirbozorgi who helped me with not only their experience and knowledge but with other aspects of my life as true friends. I would also use this opportunity to thank my good friend Carl Poirier for his friendship.

I can never be thankful enough to my dear mother, my beloved father and to my lovely sister for their sincere love. I would have been never able to be the person that I am now without you. All of you supported me wholeheartedly during all phases of my life. I love you all…

It would be impossible to finish this page without mentioning that since last year there was a guardian angel that has been taking care of me. I am grateful for every beautiful moment that you allowed me to have and for your infinite support. Kelly, whenever I remember you I have a smile on my face.

Reza Ameli,
3 Mar, 2015

1 Introduction

In virtually all animals, the control of the body and all the decision-making process is carried out by the *brain* - the center of the nervous system. Brain's functionality itself is based on the concurrent activity of billions of neurons. These neurons, although similar, are divided into smaller functional groups where each group takes care of a specific task associated with the brain [1].

When the brain (or in general the body) needs to function in a specific way or perform certain tasks, i.e. the heart needs to beat or an arm needs to be moved vertically, the corresponding functional group of neurons (cells of the nervous system) start functioning by sending electrical messages between themselves (in the group) and to other parts of the nervous system [2]. To a large extent, most of the functionality of the brain is *coded* in how and when these messages are transmitted.

Decoding the messages sourced from the neurons plays a crucially important role in understanding the functionality of the brain. The messages sent from and received by the neurons have an electrical nature and are in the form of electrical pulses with small amplitudes (between a few hundred microvolts to almost a hundred millivolts depending on where the measurement is done) [2] [3] [4]. These pulses can be captured using small sensors [4] [5]. The messages that need to be sent by the neurons are, among others, encoded in the presence/absence and also the temporal rate of these electrical pulses [6].

As mentioned before, billions of neurons constituting the brain are, from a functional point of view, divided in smaller groups where each group takes care of a certain task. Thus, when the body (hence the brain) functions in a specific way, it is expected that the corresponding group of neuron becomes more active. As an example, the *primary motor cortex* is a region in the brain that controls the several voluntary movements of different body parts [2]. Different regions in the primary motor cortex are mapped (by experiment) to the movement of different body parts. As a result, when a specific body parts like an arm or a leg is moving, we can expect that certain groups of neurons in the primary motor cortex are more active than others where activity means more transmission and reception of electrical pulses.

If, by some means, one can accurately control and study the electrical pulses that circulate between the neurons, we have in fact decoded the functionality of those neurons i.e. we have discovered the probable patterns of message transmission when the brain is functioning in a specific manner. We can take advantage of this understanding of the brain inner-working to create electronic devices that directly interact with the brain at a cellular level – *brain-machine interfaces*.

Brain-machine interfaces (BMI) or brain-computer interfaces (BCI), are devices that interact with the brain at the neural level. Figure 1 shows a high-level block diagram of such system. It can be seen that both neural stimulation and recording play an important role in this system. One of the most important goals in the field of brain-machine interfacing is the advent and development of neuroprosthetics. Neuroprosthetics aim to enhance or restore the functionality of certain impairments in the body by directly interacting with the nervous system. Examples of the most famous neuroprosthetics are cochlear implants [7], retinal implants [8] [9], and other artificial limbs [10] [11].

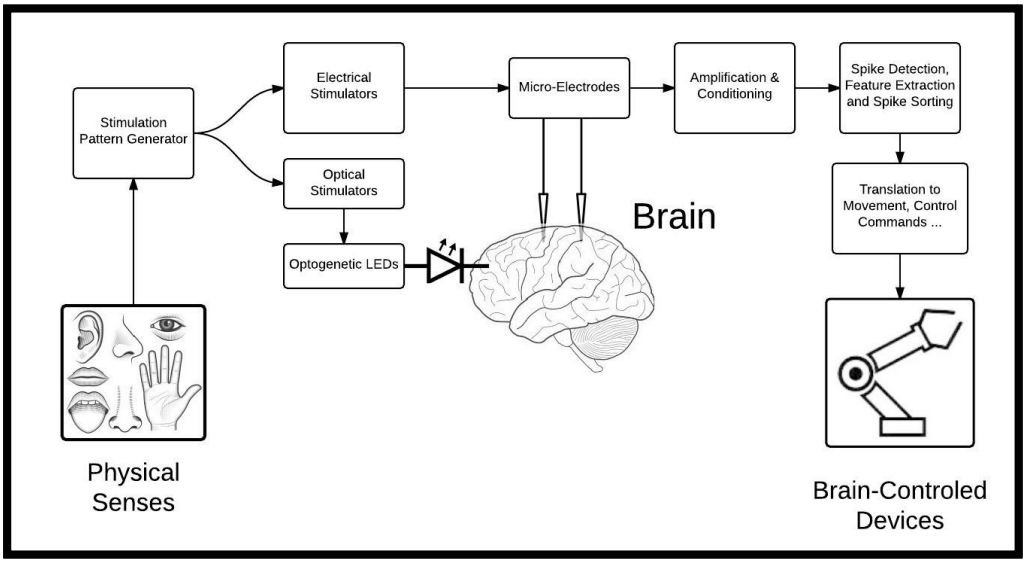


Figure 1. Block diagram of a typical BCI device with optogenetic and electrical stimulations.

Alongside with the need for neural signal acquisition, the need for neural stimulation manifests itself in the field of brain-machine interfacing as *interacting* with the nervous system requires both reception and transmission of electrical signals [12] [13]. The need for neural stimulation specially shows itself when the BCI is interacting with the sensory cortex [12] [13]. For example, if a brain-machine interface acts as an artificial robotic hand, the hand needs to receive its movement commands from the motor cortex and to send back the touch sense information back to the sensory cortex by stimulating the appropriate neurons.

1.1 Neural Recording and Stimulation

In general, acquiring neural signals is carried out via special electrodes and interfacing electrical circuitry, i.e. low-noise amplifiers and signal conditioning circuits [3] [4] [5]. Although the type, material and many other characteristics of neural sensors/electrodes are different, all electrodes will eventually be connected to some sort of amplifier and then to an analog-to-digital converter [3] [4]. On the other hand, neural stimulation is carried out traditionally in two inherently different ways: electrical or optical.

Electrical stimulation is carried out by placing microelectrodes in proximity of tissues that are receptive of electrical currents. This method of neural stimulation has been widely used in hearing aids, artificial limbs, epilepsy treatment and cluster headache treatment. The common factor of all these BCI devices is that electrical current, in a specific pattern, is generated by the microelectrodes [14]. Although electrical stimulation has been widely used, it has some inherent limitations too, including low spatial precision and possibility of inflammation and/or necrosis at the electrodes [15].

Optical stimulation, on the other hand, aims to induce neural activity via targeting neurons with light. The light source can be high-power LEDs or lasers [16] [17] [18]. Different therapeutic methods based on optical stimulation have been studied. However in this work, we focus on a specific method called *optogenetics* which is a new method for stimulating the brain functionality [19] [20] [21].

In optogenetics, light beams, emitted from LEDs or lasers, target light-sensitized neurons to activate or silence them [22] [23] [17]. One of the most important characteristics of optogenetics is high spatial and temporal accuracy, which is crucial to understanding action potential patterns [15]. Similar to electrical stimulation, optogenetics has also contributed to the understanding of different brain and cell functions and also to the understanding and treatment of certain diseases [19] [20] [21].

1.2 The Need for Freely-Behaving Animal Test Subjects

Similar to many other fields of science involving study on living bodies, understanding the inner-working of the brain requires test subjects. Test subjects in many fields are small rodents (for their availability and price) and optogenetics is not an exception [17] [24]. In this work, we aim to build an optogenetics headstage (research device mounted on the head of animals) suitable for tests on freely-behaving small rodents. The reason behind the focus on freely-moving animals is that when animals are freely behaving, certain neural activity patterns of interest will be present that so a broader spectrum of activity can be measured from or induced to the animal brain [25] [26].

1.3 Goal of This Thesis

In this thesis, we try to design and build a research tool for neuroscientists that can be used in the framework of optogenetic experiments. Basically, we aim to design a wireless optogenetic *headstage* that is capable of, simultaneously, stimulating the neurons using light and recording the electrophysiological responses (in terms of neural activity). By the term “wireless headstage”, we mean a light-weight device that can be mounted on the head of small rodents and that is not tethered to any other devices. The headstage will have embedded optical stimulation and neural recording circuitry and is capable of sending the recording neural responses back to a base station computer in real-time.

As mentioned in the previous sections, optogenetics is a powerful method for analyzing the neural circuits and as a result, taking advantage of this method requires specific tools designed for this purpose. Using research tools similar to what has been designed in this thesis, researchers can use optogenetic methods to study different disciplines such as neural circuits and neurodegenerative disorders.

We have tried to make this optogenetic research tool as close as possible to the real needs of neuroscientists. In order to do so, along with a complete literature review on the subject, we worked with a local company[†] specializing in optogenetic research tools. Thanks to this collaboration we gathered a set of realistic criteria that were deemed desirable for this type of research tool in the neuroscience community.

The mentioned characteristics include being wireless, begin light-weight and having multiple stimulation and recording channels. These characteristics make the optogenetic research tools suitable for a vast variety of optogenetic experiments where freely-moving animals can be studied in different scenarios. Chapter 0 discusses the headstage design in detail.

1.4 Contributions

The contribution of this work is a wireless optogenetic research tool that allows simultaneous multichannel recording and stimulation. This research tool is a small animal headstage that has integrated neural recording and optical stimulation and is completely wireless. In this headstage, a light-weight and small foldable PCB implements the interconnect between the (COTS) headstage components and is also the carrier of the system. Optical stimulation sources are discrete LEDs that can carry currents up to 150 mA and stimulation patterns are flexible PWM signals. Furthermore, the lifetime of experiments using this headstage is more than 3 hours. To our knowledge and at the time of this writing, this work is the first reported wireless optogenetic headstage having the mentioned capabilities.

The design of the wireless optogenetic headstage (this work) is based on the experience acquired during the design of proof-of-concept version with limited capabilities (refer to section 4.2). At the beginning stages of the design process of the new headstage, Reza Ameli (author of this thesis) worked alone on the project. However, as he approached the end of the project, two other M.Sc. students at Laval University were involved in the project. These two students were Gabriel Gagnon-Turcotte and Alireza Avakh Kisomi and they both have contributions to different parts of this project.

Gabriel Gagnon-Turcotte mostly worked on the digital, software and firmware aspects of the project. The microcontroller firmware codes (in the appendices) and the PC software that controls the headstage have

[†] Doric Lenses Inc., Québec, Canada.

been designed and/or improved by him based on the previous work of Reza Ameli. He also worked on the prototype PCB (printed circuit board) of the headstage (along with Reza Ameli and Alireza Avakh Kisomi). Besides the mentioned contributions, he worked on the Optical Stimulation Circuitry and the Analog Front End. Finally, some of the charts, plots and photos that have been used in Chapter 1 in this thesis have been designed by Gabriel. These contributions in Chapter 1 are identified in the appropriate page footer.

Alireza Avakh Kisomi worked mostly on the analog circuitry of this project. He has contributions to the Analog Front End (AFE), Power Management Unit and the Optical Stimulation circuitry. He designed different PCBs for the AFE, designed low-noise signal generators for them and characterized them. He also contributed to the design of the prototype PCB and tested the system prototype (with Reza Ameli and Gabriel Gagnon-Turcotte).

Reza Ameli worked on both the old and the new versions of the wireless headstage. He designed the first versions of the microcontroller firmware and the PC program that controls the headstage. The new firmware and the PC GUI (graphical user interface) are based on the code and/or libraries that were designed by Reza Ameli. In terms of the analog circuitry, Reza Ameli (along with Alireza Avakh Kisomi and Gabriel Gagnon-Turcotte) worked on the Analog Front End and made sure it worked properly with low levels of input signal and high levels of noise. Along with Alireza Avakh Kisomi, he also designed the Power Management Unit and its passive filter. Finally, he contributed to the design of the prototype PCB and also designed the six-layer PCB of the final headstage with EMC/EMI (electromagnetic compatibility/electromagnetic interference) considerations. Besides the mentioned contributions, Reza Ameli also worked on the specifications of the system and on how they should be realized.

1.5 Structure of This Thesis

This work presents the design process of an optogenetic headstage that is capable of stimulating neurons using light and recording the neural activity in the brain of freely moving-animals.

Chapter 1 of this thesis is dedicated to literature review and understanding the neural activity. In this chapter, action potentials being the carriers of information between neurons are introduced and discussed. Then, action potentials are treated and characterized as time series and their mathematical properties are discussed. At the end of this chapter, an overview on basics of the optogenetic technique and also a review of state of the art headstages are presented.

Chapter 0 presents an introduction to the digital signal processing of neural signals suitable for implantable devices. Topics such as detection of action potentials, feature extraction and spike sorting are discussed.

Chapter 0 has two main parts. The first part of this chapter discusses different design requirements of an optogenetic headstage system including the signal acquisition and optical stimulation requirements. These requirements include those guaranteeing signal acquisition fidelity and optical stimulation accuracy requirements. Issues that one might encounter when designing such headstage systems are also discussed in this chapter. These issues include electromagnetic compatibility (EMC), electromagnetic interference (EMI) and also low-power design issues. In the second part of Chapter 4, the details of the headstage that has been designed and the rationale behind the design are elaborated. All subsystems of the proposed headstage are discussed in detail in this chapter.

Chapter 1 presents the measured results of this research project, discusses the system performance and shows the realized headstage in detail. Physical design of the headstage is also discussed in this chapter.

And finally, the conclusion of this thesis is presented along with the appendices.

2 Literature Review on Neural Signal Recording and Headstages

2.1 Introduction

As mentioned in the previous chapter, one of the most important functionalities of neural headstages is recording the neural signals in the brain and since this recording is carried out via electrical circuits, the electrical characteristics of neural signals prove crucial in the design and test of the circuitry that amplifies and conditions the acquired signals. This chapter is devoted to understanding the neural signals *as electrical signals*. Of course, neural signals can be studied from different points of view. However, we are interested in certain of their characteristics, which allow us to design and build neural headstages.

In the first two sections of this chapter, neural signals and their electrical properties are introduced and in the third section we will briefly introduce the technique of optogenetics that is a new method for optical neural stimulation. Finally, in the last section, we will present a literature review on the state of art headstages.

One of the most important aspects of neural headstages (especially the ones that operate wirelessly) is their signal processing capabilities. Without digital signal processing, wireless neural headstages will have to transmit all digitized samples of the acquired signals to a base station, which means higher energy consumption in the headstage, higher bandwidth requirements and the need for high speed radio transceivers [27]. The mentioned characteristics are not desirable for wireless neural headstage as the limits of the energy source and the transmission rate of the RF (radio frequency) system are design bottlenecks.

Since digital signal processing capabilities (soft capabilities of headstage) can be treated separately from the actual hardware and physical features of headstages, we have dedicated a whole chapter of the thesis to neural signal processing.

2.2 Physiological Aspects of Action Potentials

Neurons are cells that constitute the nervous system. Using the nervous system, humans, among many other species, are capable of acting as intelligent entities i.e., they are able to interact with the world and think and decide. The human brain consists of approximately 10^{11} neurons and these neurons are supported by the glial cells [2]. The glial cells support the neurons in different ways including but not limited to: 1) nutrition of neurons, 2) deactivation of some neurotransmitters, 3) integration with the blood-brain barrier, 4) facilitating the action potential transmission and 5) removing cellular debris during neuronal death [2]. The complex functionality of a brain is based on neurons capability of sending and receiving messages. These messages have an electrical nature and are created in an electrochemical process that takes place in close proximity *and*

inside the neurons [2]. As a result of this mechanism, the neurons are capable of sending messages in a wave-like manner to each other. These waves of messages will eventually result into some action commanded by the brain [2]. The resulting command might be received in the brain itself or by the peripheral nervous system, which connects different components of the body to the central nervous system [2].

In the next sections of this chapter, we will describe the physiological origins and also the mathematical characteristics of action potentials. These mathematical characteristics are interesting from an electrical engineering point of view when designing headstages.

2.3 Electrochemical Process of Creating Action Potentials

As mentioned before, neurons are able to send electrical messages to each other and these electrical messages are called *action potentials*. It should be noted that neurons are similar to other cells (in terms of internal components), but they are also able to send electrical messages [2]. Also, it should be noted that there are different types of neurons; some of them act as connections between other neurons and some are able to interact with other types of cells like muscle cells. There is also one type of neuron that is able to be stimulated by different non-neural stimuli. The headstage system introduced in this work focuses mostly on the neurons that are found in the brain and transfer messages between other neurons.

A neuron as a cell has, among others, a *cell body* (also called the soma), a number of *dendrites* and an *axon*. Dendrites and axons are extensions of the cell, which receive and transmit electrical signals, respectively. Most neurons have only one axon but they can have many dendrites. Axons are the transmission medium of the action potential and tend to be long, depending on the type of the neuron. Figure 2 (derived from [28]) shows a drawing of a neuron where it can be seen that the axon is a long extension of the cell. The many dendrites of the same cell can also be seen.

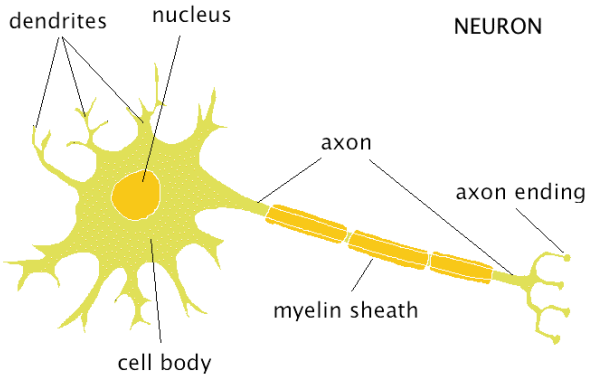


Figure 2. Sketch of a neuron (derived from [28]).

The axon of one neuron is connected to the dendrites of one or many other target cells (either neurons or other types of target tissues like muscle) [29]. So, these target cells can receive the action potentials transmitted by the first neuron. Between the axon of one neuron and the dendrites of another neuron, there is a small gap called the *synapse* where neurotransmitters travel from the axons to the dendrites and transfer the message carried by the action potential. Figure 3 (derived from [30]) shows the connection of one axon to the dendrites of another neuron.

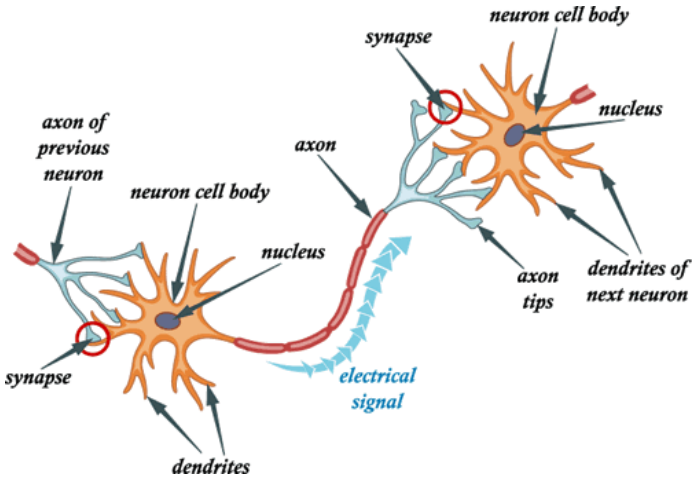


Figure 3. Connection of one neuron to another (derived from [30]).

2.4 Creation of Action Potentials

A neuron can be either at rest or can be in the process of creating an action potential. When a neuron is at rest there is a negative potential difference between the inside of its cell body and the outside of the cell where the outside of the cell has a more positive voltage. This voltage difference is due to the concentration (and also type) of ions that are present inside and outside the neuron body. The resting potential (voltage difference between inside and outside of a neuron at rest) is around -70 to -80 mV [2].

When a neuron is stimulated by an action potential, it opens its ion channels for the positive sodium ions to rush in the cell body. These ions increase the inner potential of the neuron from almost -70 mV to almost +20 mV [2]. After some time, the sodium ion channel closes and another type of ion channel opens that lets the positive potassium ion leave the neuron body; as a result, the inner potential decreases until it reaches the resting potential. This process in which the inner potential of the neuron depolarizes results in the creation of an action potential. With similar mechanisms, this sudden change in the inner potential travels through the axons and reaches another neuron where another action potential might or might not be created depending on many factors [2]. Figure 4 (derived from [31]) shows the neuron inner potential during creation of an action potential.

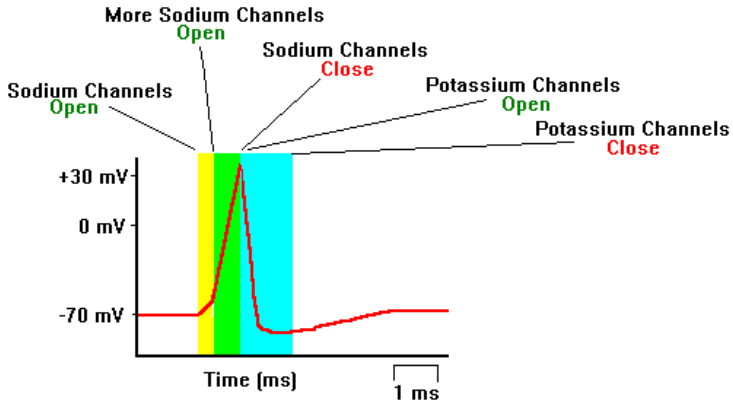


Figure 4. Inner potential of a neuron during creation of an action potential (derived from [31]).

2.5 Action Potentials and their Mathematical Characteristics

When sampled, action potentials are discrete-time signals so many mathematical properties can be attributed to them. In this section, we will introduce different mathematical characteristics of the neural signals.

2.5.1 Time-Domain Characteristics of Action Potentials

Temporal characteristics of neural signals are of significant importance as they will directly affect the design process of the brain-machine-interfaces. Although many different (statistical) time-domain characteristics can be defined for discrete-time signals, only the following temporal characteristics are mostly used for processing neural spike-trains [32] [33] [34] [27]:

1) Temporal duration (minimum, maximum and average) of individual action potentials
2) Number of action potential per second (rate)
3) Average duration of inactivity between two action potentials

2.5.1.1 Temporal Duration

In general, temporal duration of action potentials, although very close, is different for each species [29]. Different research papers have also reported average and minimum/maximum durations of action potentials for specific species based on many recordings [35].

When it comes to designing signal processing systems and algorithms, the temporal duration plays an important role since a system which is not flexible enough (in the time domain) might partially or completely lose information carried by the action potentials. Different research papers have tried to find minimum and optimum requirements for neural signal processing and also to find reasonable assumptions about the incoming action potentials trains [36]. Specifically, some papers have tried to regenerate action potential trains in a realistic way [32] [37] [38]; using these artificial but realistic spike trains, researchers can develop new

signal processing algorithms. The following table summarizes the action potential duration found in different literature.

Table 1. Action potential duration in the literature.

Publication	Average Duration (milliseconds)
[33]	2.5
[34]	2.6-3.6
[27]	1.025
[39]	2
[35]	1.46

It can be seen that the average duration of an action potential is almost 2 milliseconds and the maximum and minimum are 3.6 and 1 milliseconds.

2.5.1.2 Number of Action Potentials per Second

As the temporal duration of action potentials affects the short-term memory requirements (i.e., small buffers that only hold the data of one spike) of neural signal processing systems, the number of action potentials per second manifests its effect in the long-term memory requirements or transmission speed of such systems.

Unlike the temporal duration, the frequency (number of spikes per second) can vary considerably according to the conditions and stimulations of a neuron [35]. In fact, the frequency of occurrence of action potentials is a key factor in designing the architecture of neural signal processing systems that are low on computation and/or energy resources, especially when such systems have multiple input channels. The following table summarizes the average action potential firing rate found in different publications.

Table 2. Average Firing Rate of Action Potentials.

Publication	Average Firing Rate (Hz)
[32]	20
[33]	20
[34]	75
[40]	20-160
[39]	100
[41]	10-100

2.5.1.3 Inactive Time between Action Potentials

The third parameter that poses certain difficulties in designing low-power neural signal processing systems is the time interval in which there will be definitely no action potentials fired from a neuron. This time interval affects the data buffers in the first stages of signal processing and can be problematic in system in which the data transfer in the memory is not fast or there is not much memory. In extracellular neural recording

scenarios, there is the possibility that multiple neurons be close to the recording electrode so multiple action potentials might be captured by the electrode at the same time. In this case, it is required to separate the overlapped action potentials and some research papers are dedicated to this subject [42] [43].

The following table summarizes the average ISI (inter-spike interval) of neurons in the literature.

Table 3. Inactivity Interval of Neurons.

Publication	Average ISI (milliseconds)
[32]	2
[33]	2

2.5.2 Amplitude and Noise Characteristics of Action Potentials

Amplitude and the signal-to-noise ratio of the captured action potentials is another determining factor in designing the amplifier in the animal headstage. As mentioned before, there are different ways of obtaining neural signals from neurons [4]. However, one of the methods that is widely used and allows the researchers to carry out long experiments on the animal is extracellular recording [4]. In this method, the electrodes are placed in close proximity of the neurons(s) but not inside the neuron. Since the neuron stays intact, it does not die and the experiment can last longer compared to the intracellular recording where the electrodes are placed inside the neurons.

In extracellular recording scenarios, the amplitude of the captured action potentials is reported to be between 50 microvolts to a few millivolts. The following table summarizes the reported amplitude of action potentials in extracellular recordings.

Table 4. Action Potential Amplitude in the Literature.

Publication	Average Action Potential Amplitude (microvolts)
[44]	< 500
[27]	< 200
[45]	50-150
[3]	50-200
[4]	50-500

2.5.2.1 Sources of Noise in Neural Signals

Different sources of noise affect the quality of recorded action potentials [32]. However, unlike other signal acquisition scenarios occurring in other domains of electrical engineering, these noise sources are not all white and/or Gaussian.

In general it can be said that four different sources of noise affect the quality of a captured action potential [32]:

1) Background noise: activity of neurons that are very far from the electrode. This neural activity (action potentials) shows itself as a low-amplitude thick cloud of noise in neural recordings.
2) Activity of nearby neurons: this kind of noise is actually action potentials from relatively far neurons with amplitudes so small that it is not possible to determine the source neuron creating the action potential. The difference between this type of noise and the previous one is that, this type of noise is mostly recognizable action potentials interfering with the neural recording processing while the first type of noise is not distinguishable from random noise.
3) Thermal noise and inherent noises of electronics and the electrodes involved in the system
4) Power-line noise: 50 Hz or 60 Hz power line noise.

One of the most important aspects of the first and second sources of noise is that they share the same frequency content as the signals (action potentials) of interest. The reason is simply that these noise sources are also neurons similar to the neurons from which we would like to capture signals [32].

2.5.3 Frequency-Domain Characteristics of Action Potentials

So far we have discussed the different characteristics of action potentials in the time domain. However, the spectral content of action potentials also plays an important role in the design and quality of a headstage system.

In extracellular capturing scenarios, typically two spectrally separate signals are captured: 1) the low frequency local field potentials (LFPs) and 2) the action potentials [46] [4]. The LFPs usually occupy a bandwidth from a few Hz to almost 300 Hz [4] [32] [33], while the action potentials occupy a larger bandwidth starting from 300 Hz to almost 3000 Hz or more [4] [32] [33]. The following table summarizes the reported action potential bandwidth in the literature.

Table 5. Spectral Content of Action Potentials.

Publication	Lower Limit (Hz)	Upper Limit (Hz)
[32]	300	3000
[33]	300	6000
[34]	300	N/A
[47]	300	3000
[48]	250	5000
[49]	300	10000
[35]	300	3000

Another point worth mentioning is that the amplitude of the LFPs is much larger than that of the action potentials [4]. As a result, care must be taken when designing analog or mixed-signal systems that process the LFPs and the action potentials because the LFPs might saturate the signal processing block. As an example, an amplifier with a gain of 5000 V/V would amplify the action potentials to less than 2 volts. However, the LFPs having amplitudes in the millivolt range would saturate the amplifier when the power rail is not large enough.

In this project, we focus on capturing and processing only the action potentials. So the LFPs are filtered out in the early stages of the signal chain.

2.6 Optogenetics

Optogenetics is a neural activity control method in which light controls the activity (or lack) of genetically light-sensitized neurons [50] [22] [23] [17] [51]. In this method, different techniques in genetics, electronics and optics are combined to provide neuroscientists with a neural stimulation tool with high temporal and spatial resolution. As mentioned in the introduction, the field of optogenetics has made great contributions to the understanding of the neural circuits in the brain and also to the treatment of different diseases [50] [19] [20] [21].

In optogenetics, light-sensitized proteins (for example ChR2, ChR1, VChR1, SFOs and NpHR) are used to make neurons respond to light quickly. For example, ChR2 causes the neuron to depolarize (activate) in response to blue light and NphR causes the neuron to silence in response to yellow light [51] [50] [52]. The light source can be high-power LEDs or lasers tuned to specific wavelengths [16] [17] [18].

Optogenetic tools provide the researcher with stimulation temporal accuracies in the order of milliseconds. Researchers also need to track the effects of the stimulation by some means. One way of carrying out such readings is to monitor the neural activity using microelectrodes connected to signal-recording electronics.

In this project, an animal headstage that is capable of delivering light to light-sensitized neurons has been designed. This headstage is also capable of electrically recording the neural activity resulting from the stimulation. Optogenetic experiments can be carried out on small animal test subjects using this headstage as it has a small footprint and low weight. This headstage has two LED-based optical stimulation channels and has also two electrophysiological reading channels.

2.7 State of the Art Wireless Neural Headstages and Brain Interfacing Systems

In this section, an overview of currently available wireless animal neural research systems is presented. We focus, in particular, on the part of these systems that interfaces the brain and that are wireless. Both commercial and research headstages are presented. In general, we can divide the brain-interfacing systems into two groups: 1) systems that are based on an ASIC (application-specific integrated circuit), and 2) systems that are based on discrete electronic components.

2.7.1 Neural Interfacing Systems Based on ASICs

These systems are specifically designed to be extremely low-power, compact and efficient. A high number of neural recording/stimulation channels are also of great importance. Although this type of neural interfacing systems is not the focus of this work, a short overview of available systems is presented in Table 6.

Table 6. Comparison of ASIC-Based Brain Interfacing Systems.

Work	No. of Recording Channels	Ch. Sampling Rate (KSamples/sec.)	Bits/Sample (Stim. or Rec.)	Power Consumption (mW)
[27]	128	40	9	6
[49]	256	20	5	5.4
[44]	12	40	10	12
[41]	100	15	10	13.5
[3]	16	30	8	2.21
[53]	64	62	8	14.4
[46]	96	31	10	6.4

As can be seen in Table 6, the power consumption of ASIC-based systems is low relative to the discrete system (presented in the next section) and the number of recording/stimulation channels is relatively higher. Although ASIC-based systems can have a much superior performance compared to the discrete systems, their fabrication costs are also much higher. Furthermore, there is a clear trade-off between the system configurability, cost, complexity and time-to-market.

In the next section, we present an overview of research and commercial headstages (neural interfacing systems) that are based on discrete components.

2.7.2 Neural Interfacing Systems Based on Discrete Components

Compared to ASIC-based neural systems, headstages that are based on discrete components tend to occupy a bigger volume and to be less power-efficient. However, the configurability of these systems (in terms of upgrading the functionality) and their much lower cost make these devices much more common and desirable in the industry as well as in practical neuron-research labs.

The higher configurability of discrete headstages comes from the fact that they usually incorporate some sort of available processor (or other types of configurable devices like FPGAs) which is easy to program. Also, the manufacturing time and complexity of these headstages is lower since only discrete components need to be bought and soldered on some sort of carrier, usually a PCB in a biocompatible package. From this point on

and for brevity, the term headstage means an animal headstage based on COTS (commercial off-the-shelf) components.

In our research to identify (COTS-based) headstages we encountered some headstages that supported only one channel of data acquisition and they would transmit the acquired data using continuous FM (frequency modulation) modulation [54] [55]. However, there were other publications that would time-multiplex multiple channels of analog neural signals into one broadband channel and would then FM-modulate that channel [56]. One problem with FM modulation is that the quality of the signal that is finally received by the *base station* depends on the quality of the FM transmitter, the analog time-multiplexer (if any), antennas and all other parameters that affect analog modulation quality [54] [55] [56].

On the other hand, there are many wireless headstages that use digital modulations to transmit the acquired data back to the base station. For example, [57] and [58] use Bluetooth technology to transmit the data back to the base station and have 1 and 16 data acquisition channels, respectively. There are also headstages that use raw-data transmitters (no protocol stacks) to transmit data [17].

Recently, there has been a tendency to build light-weight and compact optical stimulation tools for optogenetics. In [59], researchers have developed a 4-by-4 array of LEDs fused with electrodes for optogenetics stimulation and neural recording. In [60], a wireless headstage with an array of optrodes and micro-LEDs has been presented. In [61], authors have designed an implantable optogenetic interface with 4 optical stimulators while in [62], a 64-channel optrode array with light-delivery system has been deign for optogenetic stimulation. Finally, in [17], the design of a wirelessly powered optogenetic headstage with a high-power LED and two recording channels has been discussed. There are also some optogenetic headstages without recording capabilities where the optical power delivered to neurons is supplied by a wireless power-delivery link [63].

Table 7 presents a comparison between currently available COTS-based headstages. In this table, some optogenetic stimulators are also shown that are not complete headstage systems but can be connected to wireless transmitters.

Different types of optogenetic and non-optogenetic headstages were presented in this review. One of the common features of all (except for [17]) optogenetic stimulators in Table 7 is their lack of high-power optical stimulation. In [63], however, high-power stimulation is possible but there is no means for neural signal recording.

Besides the research headstages that were mentioned in this section, there are also commercial headstages available for optogenetic and non-optogenetic experiments [64] [65] [66]. However, at the time of this writing, these commercial solutions did not have an equivalent for the headstage design that is presented in this work.

Table 7. Comparison of COTS-based Optogenetic and Non-optogenetic Headstages.

Work	No. of Recording Channels	No. of Stimulation Channels	Sampling Rate per Recording Channel (Samples/Sec.)	Bits/Sample (Stimulation or Recording)	Stimulation Power Consumption	Total Power Consumption
[54]	1	N/A	N/A	N/A	No Stimulation	≈3.8 mW[1]
[67]	1	1	11,7	N/A		40-120 mW
[68]	1	1	10,000	N/A		N/A
[57]	1	1	12,000	12		≈220 mW[2]
[58]	16	N/A	25,000	10		N/A
[56]	15	N/A	20,000	12		30.8 mW
[69]	32	N/A	30,000	12		142 mW
[17]	2	1 (opt. fiber)	20,000	8	20 – 380 mW	115-475 mW
[59]	16	16 (surface LED)	N/A	N/A	Minimum 3.4 mW per LED	21 mW (LEDs)
[60]	32	32 (surface LED)	N/A	N/A	Average irradiance of 1.4 mW/mm^2 per optrode tip	N/A
[61]	N/A	4 (surface LED)	N/A	N/A	No Stimulation	N/A
[62]	64	N/A	32,000	14		N/A
[63]	16	N/A	N/A	N/A		up to 2 W

[1] This value has not been mentioned in the paper but has been deduced using calculation made by author of this thesis.
[2] This value has not been mentioned in the paper but has been deduced using calculation made by author of this thesis.

3 An Introduction to Digital Signal Processing of Neural Signals

3.1 Introduction

In this chapter, we will present a short introduction to digital signal processing techniques related to neural signals that are suitable for implantable devices and neural recording embedded systems. These techniques, being *basic* processing methods, are related to finding action potentials in neural signals and classifying them according to their signal shape.

As discussed in Chapter 0, after finding the firing patterns in neural signals, we will be able to translate the neural activity (firing rate of each individual neuron) to a meaningful action, for example, moving a robotic arm. In general, an *action* that is intended by the brain is realized via changes in firing rates of different neurons. As a result, finding firing patterns of many neurons, can result in inferring the intended action [2]. In order to find firing patterns of many neurons, the fired action potential must first be detected. Then, the detected action potential must be associated with a nearby neuron according to its signal shape [36] [70]. After these two steps, the firing rate of a neuron, at a certain point in time, can be described in firings/second. In this chapter, terms *spike* and *action potential* are used interchangeably.

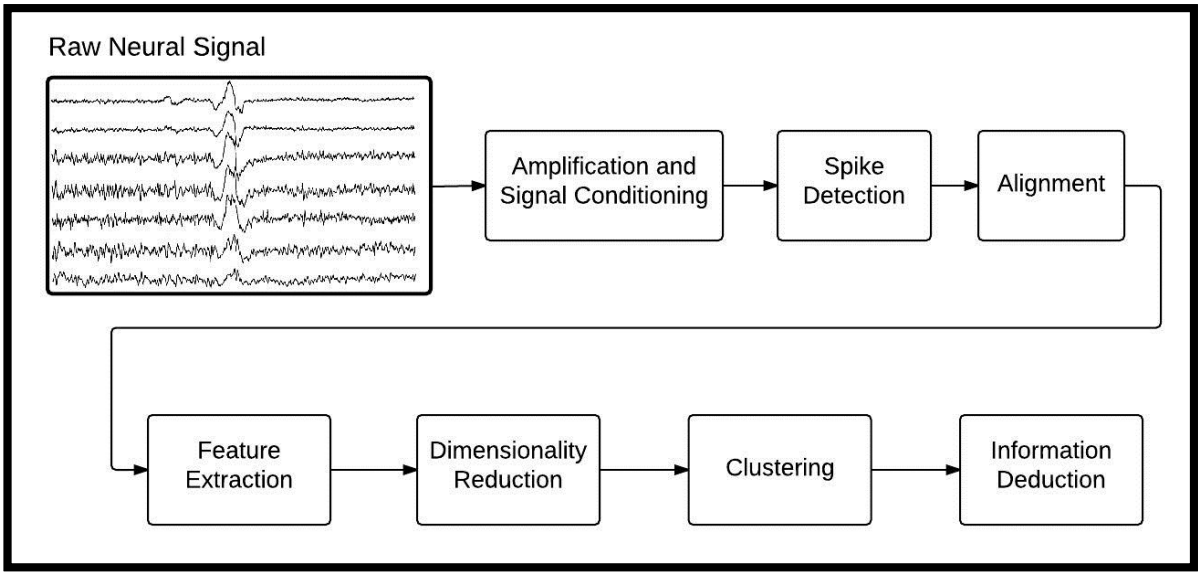


Figure 5. Overview of neural signal processing for spike sorting.

As mentioned in the previous chapter, action potentials can be recorded using electrodes. The electrodes can be placed inside the brain at proper locations to pick up the action potentials fired by one or more neurons. The (signal) shape of the action potential picked up by an electrode, depends on different parameters including

the distance between the electrode and the neuron, and also the type/shape/size of the neuron. So if there are multiple neurons in close proximity of an electrode, it can be predicted that *action potentials fired by each neuron will have a similar shape that is different from the action potentials from the other neurons* [36] [70]. The difference in action potential shapes is the basis of action potential classification, a process that is also called *spike sorting*. Figure 5 shows an overview of the required actions to sort (classify) action potentials. Spike sorting algorithms use the shape of each action potential to relate it to one neuron in the proximity of the recording electrode [36] [70].

In this chapter, we will present different steps of typical spike sorting algorithms. The algorithms reviewed in this chapter are mostly designed for small neural recording embedded systems. These systems, either based on discrete components or VLSI (very-large-scale integration) technology, usually have a limited processing power i.e., they are not high-speed as desktop computers and they have limited memory resources. The need for neural signal processing in (low-power) embedded systems and/or implantable devices comes from the fact that a real-time brain-machine interface (connected to the brain) cannot rely on offline signal processing and needs to react to neural activities in real-time.

3.2 Overview of Steps in Spike Sorting Algorithms

In order to classify (sort) the spikes according to their shapes (in the time domain), we first need to detect and extract the spikes i.e., the time-window containing the spike must be pinpointed in the neural signal [36] [3] [33] [70]. Figure 6 shows such time-window around a typical action potential. It is assumed that the neural signal is already bandpass-filtered as discussed in section 2.5.3. This bandpass-filtering removes the LFP (local field potential) component of the neural signal.

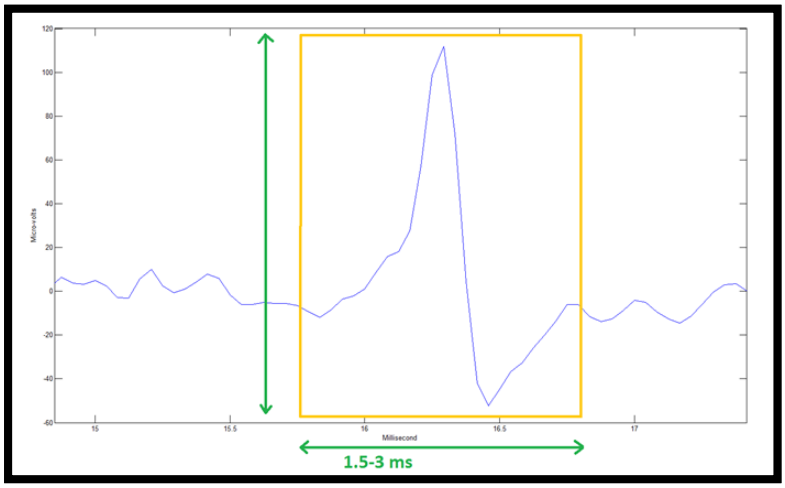


Figure 6. Time-window required to process an action potential.

After detection, spikes are usually *aligned* with respect to a certain point in their waveform. For example, spikes can be aligned in a way that their largest samples (the maximum in the waveform) would be placed at the same point if all detected spikes are overlapped [36] [70]. More details about spike detection and alignment will be provided in the subsequent sections of this chapter.

In the next step of spike sorting, the prepared (detected and aligned) spikes must be passed through a *feature extraction* algorithm where prominent characteristics of each spike are extracted. The extracted features are usually in the form of multidimensional vectors. There are various ways of extracting such vectors. The most widely used methods are reviewed in this chapter.

Finally, in the last step of spike sorting, the extracted feature vectors pointing to different points in a multidimensional space, must be *clustered* together i.e., the points that are in close proximity of each other must be assigned to the same group (i.e., the same neuron) as it is expected that they are fired from the same neuron [36] [70]. Sometimes the length of the features vectors is too long, resulting in complicated clustering (in terms of required time and memory). As a result, a *dimensionality reduction* step might precede the clustering step.

In the following sections of this chapter, different methods of spike detection, alignment and clustering are discussed in more detail.

3.3 Spike Detection

Spike detection is the process of discriminating action potentials from the background noise in a neural signal. Figure 7 shows a typical neural signal that is passed through a band-pass filter. It can be seen that the spikes can be evident as the noise level is low compared to the signal level; this is the basis of different spike detection methods [32] [36] [70].

Spike detection methods usually consist of two steps. In the first step, the neural signal is passed through some sort of mathematical operator that makes the spikes more prominent and in the second step, the output of the first step is compared with one or two thresholds [36]. In the second step, when the output of the first step is passes through the threshold(s), a recording mechanism is activated that stores the digitized samples of the action potential in some sort of memory. The number of recorded samples (i.e., the length of the time window that is saved) is usually constant [36] [70] [3] [71].

The most widely used mathematical operators for implantable devices (used in the first step of spike detection) are the absolute value and the Teager Energy Operator [72] [70]. However, the neural signal can be detected

via comparison with a threshold without any pre-emphasis on the spikes. In the following subsection, we will introduce the mentioned spike detection methods.

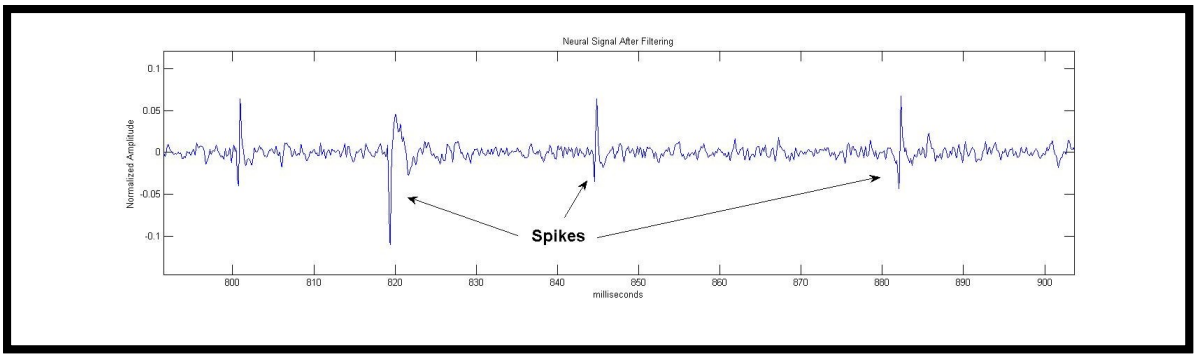


Figure 7. Typical band-pass-filtered neural signal.

3.3.1 Direct Comparison with Threshold

In this technique, the neural signal is directly compared with one or two thresholds [3] [36] [70]. This method is the simplest method of spike detection and, depending on the noise level of the recorded signal, can be the most efficient method too.

The threshold used in this method can be chosen manually or can be derived from the signal statistics [36] [70]. A widely used threshold for raw neural signals is derived from the median of the signal [33] [36] [70]:

$$Threshold = 4 * median\left\{\frac{|x(n)|}{0.6745}\right\}$$

This threshold can only be valid when all or most of the bandpass-filtered neural signal is available; as a result, real-time systems that employ this threshold should have a separate memory for storing the signal calculating the thresholds.

3.3.2 Absolute Value

As can be seen in Figure 7, depending on the relative position of the electrodes in the tissue, the action potentials might have a larger positive or negative peak. So it is more appropriate to compare the absolute value of the neural signal with a threshold [36] [73] [70]. This technique has been proven to have a better quality for spike detection than comparing the raw neural signal with a threshold [73]. The threshold used in this method can be the same threshold used for raw neural signals [36] [70].

3.3.3 Teager Energy Operator (TEO)

TEO (also known as the Nonlinear Energy Operator) is a nonlinear mathematical operator that has been devised for tone detection and FM demodulation [72] [70].

It is defined as

$$TEO\{x(n)\} = x(n)^2 - x(n+1) \times x(n-1)$$

TEO has the property of amplifying the high-frequency components of the signal. So, action potentials having higher frequency components than the background noise can result in higher values at the output of the TEO [36] [70].

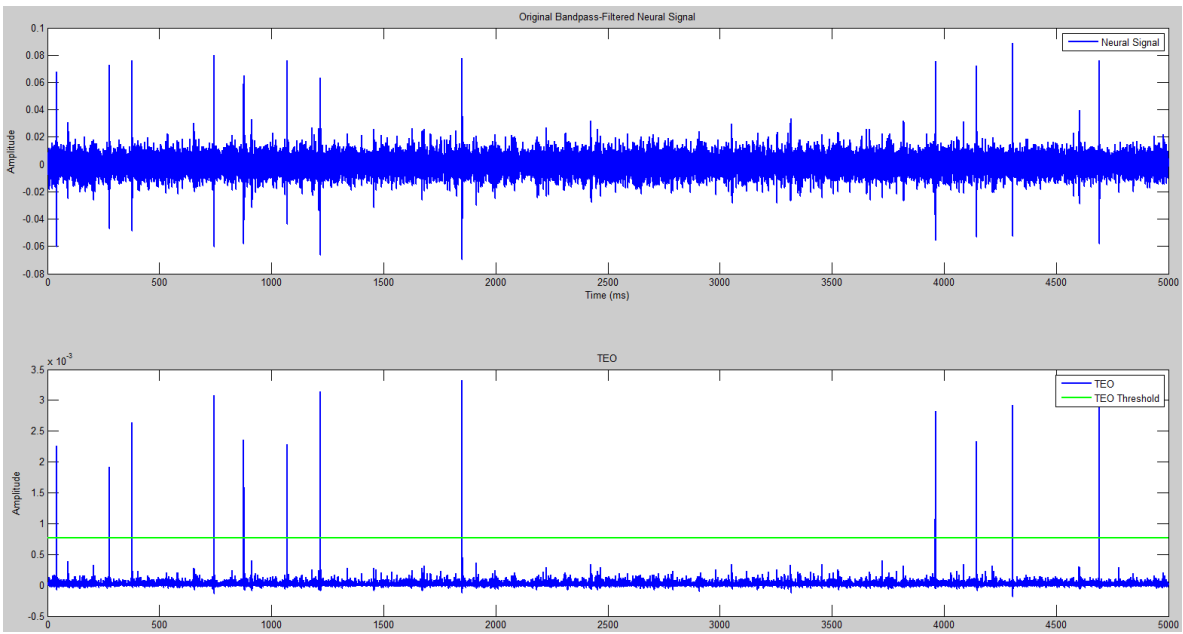


Figure 8. Neural signal passed through TEO.

Figure 8 shows a typical bandpass-filtered neural signal fed into the TEO and the output of the TEO. It can be seen that TEO emphasizes action potentials. Similar to previous methods, in order to detect spikes, the outputs of the TEO are compared with a threshold. This threshold can be derived from the average value of the TEO output because TEO emphasizes the spikes and deemphasizes the background noise—making their weight in averaging almost negligible [36] [70]. More formally, the TEO threshold can be stated as:

$$TEO\ Threshold = C \times \frac{1}{N}\sum_{i=1}^{N} TEO\{x(n)\}$$

where C is constant.

The fact that TEO threshold can be derived from averaging rather than calculating a median, makes threshold calculation much easier than the case of absolute value as calculating the mean value does not need large memories.

Besides the classical TEO, there is also a variant of the TEO called the k-TEO [74]. This operator is similar to the TEO, however, instead of delays of one sample, delays of k samples are used. K-TEO is defined as:

$$k\text{-}TEO\{x(n)\} = x(n)^2 - x(n+k) \times x(n-k)$$

It can be shown that k-TEO can be tuned, using the value of k, to detect action potentials that are wider. Also, [74] shows the application of a bank of k-TEO operators for spike detection.

3.3.4 Comparison of Spike Detection Algorithms

Different spike detection algorithms have been introduced in the previous subsections. Since these algorithms can be considered as binary classification tests, we can measure their performance using ROC (receiver operating characteristic) curves [36] [70]. The ROC curve shows the true positive rate as a function of false positive rate [75]. In general, it is desirable to have more true positive rate and less false positive rate. A comparison presented in [36] shows that the absolute value and the TEO have very close characteristics in the ROC curve. However the authors of this paper mention that, by considering the choice probabilities and the resilience to noise, TEO outperforms the absolute value. It should also be mentioned that the implementation cost in terms of operations or chip area of TEO is higher than the absolute value method [36] [70].

3.4 Spike Alignment

After detection, the spikes need to be aligned in a way that their differences/similarities would be as prominent as possible. This task is called alignment and it is assumed that all detected spikes occupy a constant number of samples i.e., constant temporal length.

Two spike alignment methods are mostly used [36] [70]:

1) Alignment with respect to sample with the maximum value: in this method, all spike waveforms are arranged in a way that their sample, having the maximum value, is at a certain point in time when all spike waveforms are overlapped.
2) Alignment with respect to the maximum derivative: in this method, all spike waveforms are arranged in a way that their sample, having the maximum slope (with respect to the previous sample), is at a constant point in time when all spike waveforms are overlapped.

Figure 9 and Figure 10 show spike alignment using the maximum value and maximum slope, respectively. In Figure 9, there are two different spike groups (having different shapes) and it can be seen that all spikes are *pinned* at the maximum value. On the other hand in Figure 10, three different groups of spikes are pinned at the point where they have the maximum slope (positive or negative).

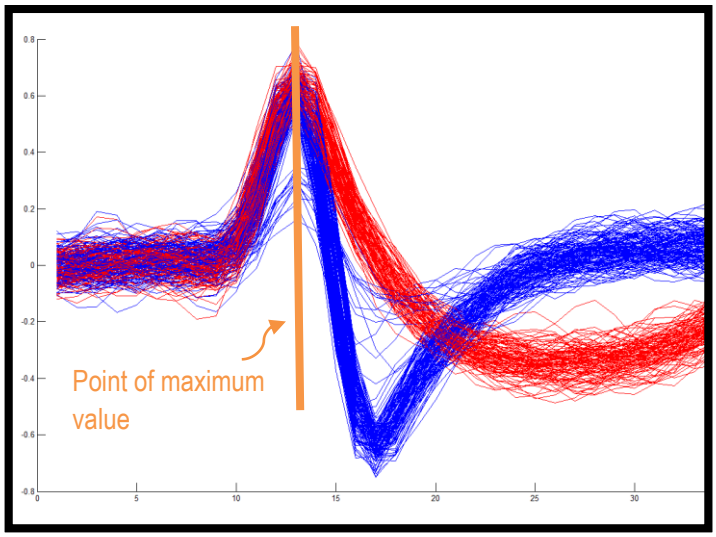


Figure 9. Spike alignment using maximum value.

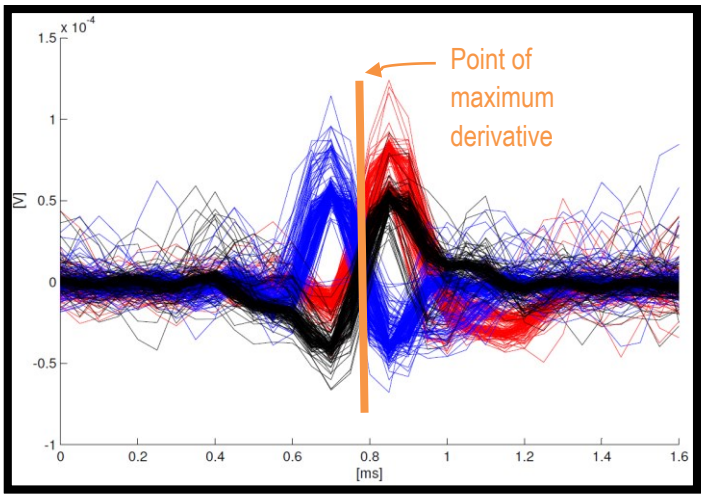


Figure 10. Spike alignment using the maximum slope.

3.5 Feature Extraction

As evident in Figure 9 and Figure 10, spike alignment maximizes the differences between the action potentials. After alignment, a *feature extraction* algorithm is needed to convert the shape of each spike to an *n*-dimensional vector. It should be noted that action potentials, being vectors of real numbers, can be already

considered as points in the space. However feature extraction algorithms tends to elaborate the prominent spike features using less number of samples.

The most widely used feature extraction methods are Principal Component Analysis (PCA), Discrete Wavelet Transform (DWT), Discrete Derivatives (DD) and Integral Transform (IT) [36] [70]. In the following sections, each feature extraction method is elaborated.

3.5.1 Principal Component Analysis

PCA is a widely used method that has been used numerous times in spike sorting applications. PCA tries to find an orthogonal basis for the data (set of all detected spikes) whose variations are maximal. This orthogonal basis can be derived via eigenvalue decomposition of the covariance matrix [36] [70] [76].

After finding the principal components, each spike can be expressed in the new basis as a set a values c_i:

$$c_i = \sum_{n=1}^{N} PC_i(n) \times s(n)$$

where $s(n)$, the spike, is a vector, $PC_i(n)$ is the ith principal component and N is the number of elements in the spike vector [70].

The c_i vector or a subset of its elements can be used for *clustering* as most of the data variations are represented in the first few principal components [36] [70].

Although PCA has been widely used as a feature extraction method, it has the disadvantage that it requires relatively many multiplications and additions making it inappropriate for real-time feature extraction in embedded systems with limits on computation power.

3.5.2 Discrete Wavelet Transform

DWT, similar to PCA, is used to derive a set of coefficients from the data that represent the differences in the data more clearly [36] [70]. DWT coefficients are derived using

$$DWT(u,j) = \sum_{n=-\infty}^{+\infty} s(n) \times \frac{\Psi(\frac{n-u}{2^j})}{2^{j/2}}$$

where u is the *translation* parameter and 2^j is the *scaling* parameter with j an integer.

The Ψ function is the mother wavelet function that is different depending on the wavelet family used for the transform. The DWT operation can be implemented as a filter bank and depending on the wavelet family, the DWT might involve many multiplication/addition operations.

3.5.3 Discrete Derivatives

Discrete Derivatives (DD) is a low-complexity method similar to a simplified version of DWT [36] [70]. It calculates the slope between the nth samples and the $(n-d)^{th}$ sample:

$$DD_d(n) = x(n) - x(n-d)$$

3.5.4 Integral Transform

The Integral Transform (IT) is another low-complexity method useful for real-time feature extraction that produces a feature vector of two elements. It calculates the average nonnegative phase and the average negative phase of the action potential waveform:

$$\begin{cases} IT_P = \dfrac{1}{N_P}\sum s(n) & \text{where } s(n) \geq 0 \\ IT_N = \dfrac{1}{N_N}\sum s(n) & \text{where } s(n) < 0 \end{cases}$$

where N_P and N_N are the number of nonnegative and negative samples of the action potential.

The feature vector will be $[IT_P\ IT_N]$, resulting in simple two-dimensional clustering.

3.5.5 Comparison of Feature Extraction Algorithms

In terms of implementation costs, the PCA and DWT methods are an order of magnitude more complex than the IT and DD methods. This is due to their memory and computation (multiplications and additions) requirements [36]. In [36] and [70], the accuracy of these feature extraction algorithms has been plotted versus their computational cost and it has been deduced that since the DD algorithm lies at the knee point of the curve it has the best accuracy-complexity tradeoff for small embedded or implantable systems.

3.6 Data Clustering Algorithms

The last step of spike sorting, also being the most complex one, is clustering the feature vectors. It should be noted that the clustering algorithms presented in this section consider the feature vectors as points in a multi-dimensional space.

Before talking about the details of clustering algorithms, it should be mentioned that there is another step, usually taken in spike sorting algorithms, that is not introduced in this chapter. This step consists of reducing the number of elements in feature vectors also known as *dimensionality reduction* [36] [70]. There are various ways to reduce the number of elements in features vectors [70] including uniform resampling, Lilliefors Test [77] and Hartigan's Dip Test [78].

Uniform resampling consists of keeping only one element out of each *k* elements in the feature vectors. However, the other two tests try to find multimodality in the elements of the feature vectors and as a result, they are more efficient. Uniform resampling can be easily implemented in low-resource embedded systems. However, the two other tests require large memories and high processing speeds, which is not suitable for low-power neural recording applications.

In the following subsections, we will present the most widely-used data clustering algorithms. It should be noted that these algorithms are more power/memory-hungry than the algorithms involved in spike detection, alignment and feature extraction. Also, they are usually iterative algorithms with an unknown number of iterations.

3.6.1 K-Means Algorithm

The k-Means algorithm [79] classifies the data points into *k* different groups where *k* is provided by the user. So, it is a supervised method (i.e., it requires manually selected parameters). This algorithm starts by randomly defining *k* cluster centroids and then, through many iterations, it recalculates the centroids by measuring the Euclidean distance between the centroids and the data points.

3.6.2 Valley Seeking Algorithm

This algorithm is based on calculating the normalized density derivatives (NDD) and finding the peaks of these functions [80]. It has the advantage of being nonparametric and unsupervised. However, its disadvantage is requiring relatively large processing power making it unsuitable for implantable applications [70].

3.6.3 Superparamagnetic Clustering

Superparamagnetic Clustering (SPC) is a clustering method inspired from the physical properties of the inhomogeneous ferromagnetic model [33] [81]. This clustering method, being unsupervised and accurate, has been used by [33] along with DWT for spike sorting. However, similar to many other clustering algorithms, this algorithm requires a large processing power.

3.6.4 Osort

This method [82], devised by neuroscientists, is both unsupervised and can be implemented on relatively small embedded systems. In Osort, the first data point (the first spike waveform) becomes its own cluster. For the following spikes, the distance (usually Euclidean) to all cluster centroids is computed and the minimum distance is considered. If this minimum distance is less than the merging threshold, the spike is added to the nearest cluster and the cluster centroid is recalculated, otherwise a new cluster is generated. While processing the incoming spikes, if the distance between two clusters becomes less than a certain value, the two clusters are merged.

3.6.5 Comparison of Clustering Algorithms

Table 8 (derived from [70]) shows the tradeoffs between different clustering algorithms. It can be seen that Osort provides the best tradeoff between implementation complexity, speed and need for supervision.

In this chapter, only the clustering *algorithms* were discussed. However, it should be mentioned that manual spike sorting, based on the visual feedback from the extracted features, can also be a viable solution. But, manual clustering cannot be real-time, hence not implantable in neural recording embedded systems.

Table 8. Comparison of clustering algorithms.

	Manual	**k-Means**	**Valley Seeking**	**SPC**	**Osort**
Nonparametric	NO	NO	YES	YES	NO
Unsupervised	NO	NO	YES	YES	YES
Real-Time	NO	NO	NO	NO	YES
Adaptive	NO	NO	NO	NO	YES
Complexity	-	LOW	HIGH	HIGH	LOW

3.7 Some Notes on Implementing DSP Algorithms

Some of the algorithms that were introduced in this chapter can be easily ported to low-power microcontrollers or FPGAs for real-time implementations. In order to implement such algorithms in digital systems the following methodology is suggested:

1) A suitable algorithm targeting low-power applications must be chosen. This choice can be affected by many factors including the digital processor (i.e., an FPGA or a microcontroller) that is going to be used, the available clock frequencies, the power budget, etc… Not all algorithms can be implemented in all signal processing systems.

2) The algorithm must be implemented and tested using floating-point calculations (for example in MATLAB) on signal obtained from signal banks. This allows the designer to verify the correctness of the algorithms rather than the implementation. Also, testing algorithms in environments like MATLAB gives the designer some insight into the inner-working of the algorithm. At this stage, the designer must make sure that the chosen algorithm is capable of processing the signal as desired on many different signals of the same family (for example on many different ECG signals from different test subjects).

3) Based on the architecture of the processor or the limitations of the FPGA that is going to be used, the designer must now simulate the algorithm with limited precision i.e., in fixed-point. This allows the designer to investigate the effects of limited precision on the algorithm. The effects of fixed-point calculations can show themselves in many different ways including the frequency response of filters or the quality of the signal after processing. This step can also be done in MATLAB.

4) After making sure that the algorithm work perfectly in fixed-point simulations, one can obtain real-life signals from the system (that is going to be designed) and test them using the fixed-point algorithms that were developed. For example, if ECG signals are to be compressed in the target systems, the designer can obtain real ECG signals using the same A/D that is going to be used later and test the compression algorithm using fixed-point calculations in MATLAB.

5) After the mentioned tests, the designer must implemented the algorithm in real-time in the target system. The way this step is done can vary from one system to another. Basically, the engineer must make sure that the delays caused by processing the signals/samples are lower than the required system response times. In the case of microcontroller, DMAs and interrupts can help a lot with reducing the delays of signal processing and in the case of FPGAs, pipelined architectures can be used.

In [83] authors present an implementation of a spike detection algorithm based on the absolute value on the hardware that has been developed in this project. This paper is an example of the methodology that is mentioned above.

3.8 Conclusion

In this chapter, a short review on one class of signal processing techniques for neural signals was presented. This class of neural signal processing is about detecting action potentials and associating them to neurons when multiple neurons are in close proximity of a recording electrode.

Other types of signal processing can also be carried on recorded neural signals, most notably action potential compression. Action potential compression allows the neural headstage or the implantable device to postpone the spike sorting to a later time in a high-speed computer while keeping the spike waveform. This results in relatively less complexity in the headstage firmware. However, even in the case of action potential compression, the need for spike detection exists.

In the next chapter, we will present the details of the headstage design.

4 System Design of the Optogenetic Headstage

4.1 Introduction

In optogenetic stimulation, the light source is usually tuned to a specific wavelength — usually blue at 473 nm [16] [17] [18] . A headstage system requires small size and weight, long life time and physical robustness. High wireless range and high number of stimulation/recording channels are also desirable characteristics as they allow carrying out optogenetic experiments on freely-moving small rodents, conveniently. Having simultaneous optical stimulation and bioelectrical recording at the same headstage would greatly enhance research capability in this area. Also, having the option to support multiple wavelengths would be a plus. However, currently there are no wireless systems available to optically stimulate the brain cells and record the neural response in real-time in multiple independent channels. Thus, the goal of this work is to realize such a research tool.

As research objectives, this project aims to design and fabricate a wireless optogenetic headstage that provides simultaneous optical stimulation and bioelectrical recording inside a wireless device operating in real-time. Moreover, it is desired that a multichannel wireless optogenetic headstage with simultaneous stimulation and neural recording be designed that allows optogenetic experiments on small freely moving animals. It is also highly beneficial that multiple stimulation wavelengths be supported and the size and volume of the device be as small as possible.

In general, wireless headstages are installed on the head of an animal (usually a small rodent) in a surgery procedure and then the headstage will be controlled from a base station where the neuroscientist can determine the stimulation patterns and visualize the acquired signals. In experiment scenarios where animals are freely moving, the animal is usually kept in a cage and its physical behavior is also monitored. Another approach to monitor an animal's neural response to stimulation is to keep the animal still while the experiment is taking place. Wireless headstages are especially useful for experiments on freely-moving animals.

Neural signal acquisition is carried out by placing microelectrodes inside the brain of an animal; then the acquired signal is amplified and filtered using specialized electronics. This analog signal chain, also called the analog front end (AFE), must be as low-noise as possible because the neural signal that is picked up by the electrodes has a low amplitude, comparable to the noise RMS (root mean square) value. The AFE must also be able to separate the high-frequency content of the neural signal (action potentials) from the low-frequency component.

After analog signal processing and digitization, some sort of digital signal processor will process the neural signal samples. This processing can be either 1) directly delivering the digitized samples to the radio transmitter or 2) performing a specific signal processing algorithm on the data prior to transmission. Different types of processing can be performed on the acquired signals, like action potential (spike) detection with or without spike sorting and signal compression.

As discussed in previous chapters, neural stimulation can have either an electrical or an optical nature. For electrical stimulation, microelectrodes can be used while for optical stimulation (including optogenetics), small optical fibers can be placed inside the brain of the animal. The optical fibers or the microelectrodes are connected to their appropriate *drivers* where the stimulation waveforms are generated.

Headstages that perform different tasks, especially signal processing, require some sort of signal processing unit to control the activity of different hardware and software components. This control unit can be a microcontroller, a DSP (digital signal processor), an FPGA (field-programmable gate array) or a combination of these. Low power consumption and real-time responses are necessary for such processing units. Finally, in cooperation with some analog circuitry, the processing unit is also usually responsible for generating the stimulation patterns.

For wireless headstages, there is also a need for wireless data transceivers that are used to transfer acquired neural signals back to a base station where the data can be viewed or analyzed. In recent headstages where multiple neural signal acquisitions are present (refer to Chapter 2), digital modulation schemes such as FSK (frequency-shift keying) or GFSK (Gaussian frequency-shift keying) are very popular for their ease of implementation [17] [69] [84].

Last but not least, the power supply circuitry of the wireless headstage can be a challenge in some scenarios where power is limited. Since headstages are mixed-signal systems, multiple low-noise power rails might be required. On the other hand, depending on the neural stimulation type, some of these power rails might experience heavy current discharge in short time intervals which leads to presence/conduction of noise on other power rails.

4.2 Design Methodology

As mentioned previously, the wireless optogenetic headstage is a research tool that helps neuroscientists to study the brain behavior in different experiments and our goal in this thesis is to design a novel wireless headstage that allows simultaneous real-time multichannel optogenetic stimulation and neural recording, which is not available so far. The focus on real-time and multichannel stimulation and recording comes from the fact

that real-time neural recording is required to investigate the effects of millisecond-scale optical stimulation and, multiple channels of recording and stimulation allow simultaneous monitoring of multiple brain regions.

In order to design this research tool, we initially designed a proof-of-concept wireless headstage and based on the experience gained during the design process, we designed the (final) multichannel wireless optogenetic headstage.

In both headstages, we opted for the following innovative approach to design and fabricate the devices:

- Providing means for electrophysiological recording and optical stimulation in the same device: it is necessary that both stimulation and recording be present in the same headstage as it is required to investigate the effects of stimulation using real-time recording (refer to sections 2.2, 2.5.1 and 2.6).
- Using inexpensive, miniature and light-weight COTS components to design a headstage suitable for small rodents: this allows having a low-cost light-weight headstage that can be easily used by neuroscientists.
- Using PCBs as component carrier and chassis for the headstage: this removes the need to have bulky containers to encapsulate the components of the headstage and results in lighter devices suitable for experiments on small rodents (refer to section 1.2).
- Incorporating high-current discrete LEDs as the optogenetic light source: high-current LEDs allow deeper optogenetic stimulation as the optical loss in the brain tissues decreases the received optical power in deeper layers (refer to section 2.6).
- Designing the analog amplifiers based on low-noise COTS to acquire neural signals: as mentioned in previous chapters, neural signals are in order of tens of microvolts and require very low-noise analog recovery (refer to section 2.5). High-quality analog recovery is crucial to further steps of the analysis of neural signals.
- Using wireless transceivers to transfer the acquired neural data to the base station: when a headstage is untethered, it can be easily used in experiments involving freely-moving animals which is the goal of this work (refer to section 1.2).
- Using low-power microcontrollers to control the headstage subsystems.
- Using reliable power sources to supply the stimulation LED and the rest of the system with energy.

In the next two subsections, we will present the proof-of-concept headstage and the final design in more detail.

4.2.1 Proof of Concept

The headstage that has been designed as the proof of concept was based on one stimulation LED and two neural recording channels [17]. This headstage was wirelessly powered using inductive links† [85] and required power-delivery chambers to operate. Also, its physical design was based on 3 (rigid) PCBs that were stacked using board-to-board connectors. The following list summarizes the characteristics of this headstage:

- Fabrication based on COTS components
- Light-weight, small and wireless headstage based on three stacked rigid PCBs
- One 100 mA optogenetic stimulation LED
- Two low-noise neural recording channels
- Microcontroller with integrated RF transceiver to control the system and transmit the data to the base station
- Inductive power chamber as power source

Different details about this headstage can be found in [17] and are also summarized in Table 9.

Table 9. Summary of the performance of the first wireless optogenetic headstage.

Parameter	Value
Weight	7.4 grams
Dimensions	15×25×17 mm
Supply Voltage	3.3 V
Readout circuitry Power Consumption	3.73 mW
PMU Power Consumption	16.54 mW
RF Microcontroller Power Consumption	74.25 mW
LED Power Consumption (Constant Mode)	(20 - 380) mW
Total power consumption (Constant Mode)	(114.52 – 474.52) mW
RF Operating Frequency	868 MHz
RF Output Power	0 dBm
Data Rate	320 kb/sec
Power Delivery Architecture	4-Coil Inductive Link
Power carrier Frequency	1 MHz
Power Transmission Range	< 7 cm
Number of Recording Channels	2
Input-Referred Noise	1 µVrms
Sampling Frequency (per Channel)	20,000 Sample/Second
ADC Precision	8 bits
Readout Interface Gain	40 to 60 dB
Readout Interface CMRR	> 100 dB
Readout Interface Bandwidth	100 Hz to 10 KHz

† The work on the inductive power delivery link (and some parts of the analog circuitry) has been done by Abdollah Mirbozorgi, PhD student at Laval University, 2013.

Figure 11 and Figure 12 show the physical design of this headstage. It can be seen that the three stacked PCBs are connected together using board-to-board connectors and that the electrodes are wires alongside the optical fiber. The power receiving coil is also sandwiched between two of the PCBs.

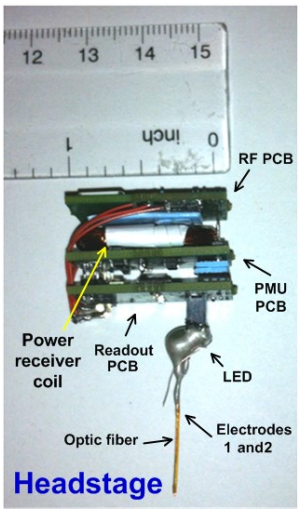


Figure 11. First version of the optogenetic headstage: stacked PCBs, power-receiving coil and the optical fiber.

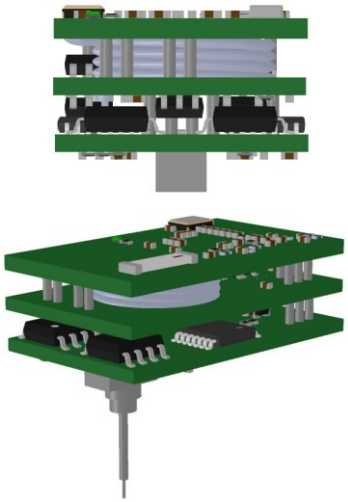


Figure 12. 3D models of the headstage: Side view with the power-receiving coil (top). Complete model with optical fiber (bottom).

4.2.2 Issues, Solutions and Design Approach

Although the proof-of-concept version of the optogenetic headstage satisfied many different requirements of optogenetic experiments, it had shortcomings such as a large size and large weight, and a lack of multichannel capability, which are essential characteristics for conducting practical experiments with freely moving animals. As a result, we have devised an original approach, which will be detailed in the following paragraphs, to design an advanced wireless headstage (detailed in Section 4.3) addressing the mentioned shortcomings. The following list contains different shortcomings and issues of the proof of concept as well as the improvements that were made in the new headstage. The solutions to the mentioned issues comprise our approach to the design of the new headstage:

1) **Issue**: The batteryless nature of the headstage and the fact that the power delivery distance (see [17]) was almost 7 cm resulted in certain limitations in some experiments. For example, if a relatively large rodent decides to stand on its feet it was possible that the headstage did not receive enough power to continue working.

   Solution: In the new design, a battery is used instead of wireless power delivery link. The system is able to work (using the battery) more than 3 hours.

2) **Issue**: The design of the proof-of-concept headstage was based on three stacked PCBs. The non-monolithic design of the first headstage resulted in higher volume and more difficulty in debugging and maintenance.

   Solution: The new design benefits from the rigid-flex PCB fabrication technology. This means that different *rigid* PCBs are connected to each other using *flexible* PCBs. Taking this approach, completely eliminates the need for board-to-board connectors and saves space and results in a robust monolithic PCB design. Also, the new headstage's dimensions will be less than 20×20×20 mm^3 and its weight will be less than 7 grams. The size and the weight of this headstage address the fact that a laboratory mouse (test subject for many different types of experiments) does not usually weigh more than 30 grams [86]. As a result, in order for the mouse to be able to freely move, the headstage weight/size must be as low as possible.

3) **Issue**: There was only one stimulation LED in the proof-of-concept headstage, which resulted in experiments with less degrees of freedom. Furthermore, the maximum current in the LED was limited to 100 mA.

   Solution: The new headstage benefits from two optical stimulation channels where each of these channels is a high-power LED (LB G6SP [87] or LY G6SP [88] by OSRAM Opto Semiconductors [89]) that will be driven by up to 150 milliamps of electrical current. At any given time only one of the LEDs will be active and the optical power generated in the LED will be delivered to neurons via optical fibers.

   Similar to the previous headstage, there will be two neural recording channels where each of these recording channels is connected to a low-noise amplifier and is properly filtered. The filtering that is done in the analog circuitry is bandpass and allows frequency content between almost 300 Hz to 6000 Hz to pass. After amplification and filtering, each channel is digitized at a rate of 20,000 samples/second. The analog front is low-noise enough to be able to extract action potentials with amplitudes as low as 10 microvolts.

   Being equipped with two reading channels as well as two stimulation channels allows simultaneous excitation and monitoring of multiple brain regions. This results in more flexible optogenetic experiments as there will be less need for changing the electrodes and/or optical fiber sites. As mentioned in chapters 0 and 1, different brain regions are dedicated to different tasks so multiple stimulation/reading channels results in more flexibility in studying the brain activity.

   The high current through the LED compensates for the losses of optical fibers and their connections. Depending on the nature of the experiment, this current flow and the resulting optical power might be

too much [59]. However, the high irradiance that flows out of the optical fiber allows stimulation of deeper brain areas.

4) **Issue**: The RF center frequency of the transmitter was 868 MHz in the proof-of-concept headstage. Optimum antennas at this frequency are usually much larger than the largest dimensions of the headstage. Since large antennas cannot be used with the headstage, the transmission distance of the headstage decreases.

   Solution: The RF transceiver in the new headstage has a center frequency of 2.4 GHz, which results in much smaller and more efficient antennas compared to the previous headstage. Also, the new RF transceiver is capable of transmitting data up to 720 kb/sec. This bit rate allows transmitting other information as well as the neural data. For example, the outputs of temperature sensors or accelerometers can be sent alongside with the neural data.

5) **Issue**: The proof-of-concept headstage had optical stimulation patterns in the form of PWM (pulse width modulation) waves. However, the transitions between the high and the low states were not fast enough compared to the time scales of the waveforms.

   Solution: The stimulation patterns that were opted for this headstage as well as the previous one are PWM signals. Since the optogenetic stimulations and responses happen in millisecond scales (see chapters 0 and 1), the stimulation and the recording of these events must be real-time enough to capture all information. Particularly for the stimulation, the state transition in the PWM signal must be much faster than the width of the pulse. This requirement is addressed in the new design of the headstage. In essence, the optical stimulation pattern of the LEDs will be a PWM signal with a frequency between 1 Hz to 100 Hz and the duty cycle will be between 0.1 % and 10 %. When the pulse is active, a current of 150 milliamps will flow into the active LED.

As mentioned above, the experience that was gained while designing the proof of concept, was leveraged to implement an advanced multichannel optogenetic headstage. The following list summarizes the approach to design the new multichannel optogenetic headstage being the subject of this project:

- Foldable rigid-flex PCB carrier with light-weight COTS components: the foldable PCB alleviates the need for board-to-board connectors and results in lower weight and size of the system which is necessary for experiments on freely moving small animals.
- Two 150 mA optogenetic stimulation LEDs: having two stimulation LEDs allow neuroscientists to monitor two brain regions simultaneously. This results not only in simultaneous study of two brain

regions but removes the need of performing two surgeries to implant the optical fibers into two brain regions of two different animals.

- Two low-noise neural recording channels: as mentioned previously, very low-noise amplifiers are required to extract the microvolt-scale neural signals from the environment noise and the quality of analog signal acquisitions affects the rest of the signal processing and analysis steps.
- Low power MSP430 microcontroller to control the system: this microcontroller has all the required capabilities to control the system, digitize the neural signals and send the data to the base station. It is also very low-power.
- Separate high-bit-rate RF transceiver capable of using small chip antennas: using an RF transceiver operating at 2.4 GHz results in more flexibility in PCB layout design as smaller antennas are used. Also the higher bit-rate of this transceiver allows transmission of different data from different sensors simultaneously, dynamically changing the stimulation pattern and results in lower power consumption.
- Small Li-Ion battery as power source: this battery allows continuous work for more than three hours without need to change batteries. Also the small form factor of the battery does not add considerable weight to the system.
- Circuitry to generate sharp optical stimulation patterns: as action potentials occur in millisecond time scales, the corresponding stimulation must also have a fine time resolution (refer to section 2.6).

4.3 The Design of the Multichannel Wireless Optogenetic Headstage

In this section, the details of the advanced headstage design are presented. At first, a short overview of the design is presented and then each component of the headstage is discussed in detail. The main novelty of this work is gathering multiple stimulations and recording channels in a small and robust wireless headstage that can be used in optogenetic experiments involving small freely moving rodents. In the next section, we will present the design details of the multichannel optogenetic headstage.

The proposed headstage system includes the following main blocks (subsystems):

1) Analog Front-End (AFE)
2) Optical Stimulation Circuitry
3) Power Management Unit (PMU)
4) Microcontroller Unit (MCU)
5) Digital Wireless Transceiver

In the AFE, bandpass filters keep the useful frequency content between almost 300 Hz to 6600 Hz; this band contains the action potentials and the LFPs. The AFE benefits from high common-mode rejection ratio (CMRR) and power supply rejection ratio (PSRR); both of these characteristics are necessary for the AFE operation in the presence of different types of noise. The high CMRR removes the 50/60 Hz power-line noise and other types of common-mode noise while the high PSRR removes the power supply fluctuations that can be conducted to the AFE inputs.

The optical stimulation circuitry generates the required PWM waveforms that will be applied to the stimulation LED terminals. This subsystem receives a digital PWM signal from the MCU and converts it to electrical current in the LED.

The PMU is responsible for providing other subsystems with appropriate power rails. These power rails are required to be as low-noise as possible because any noise on the power rails directly affects the quality of the acquired signals. In order to remove the high-frequency noise on the power rails, the PMU incorporates a low-pass power supply filter. Details of this filter are provided in the following sections.

The MCU is a low-power microcontroller from the MSP430 family of microcontroller from Texas Instruments. This microcontroller is responsible for delivering the data to the radio transceiver and providing the stimulation LED drivers (optical drivers) with appropriate PWM signals. Furthermore, at the beginning of an experiment, the MCU receives the experiment parameters such as stimulation patterns and the stimulation duration from the base station.

Figure 13 shows the block diagram of the system. It can be seen that the MCU controls all other subsystems of the headstage. Four microelectrode will be connected to the brain tissue; one of them brings the brain tissues to an appropriate common-mode voltage and the three other microelectrodes are the inputs of the AFE.

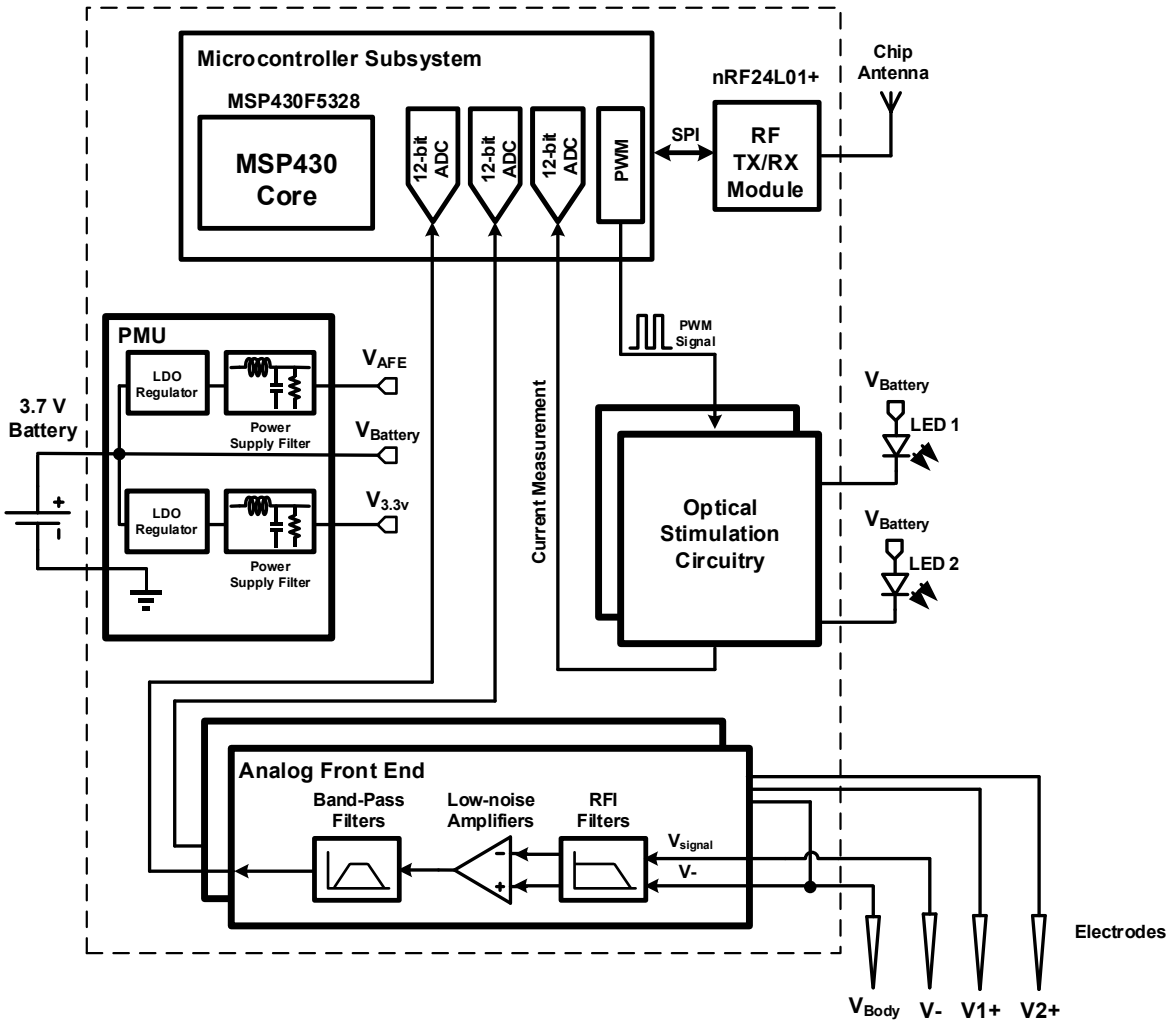


Figure 13. Headstage block diagram.

In the following sections of this chapter we will discuss each subsystem in detail.

4.4 Analog Front-End (AFE)

The AFE, being designed for extracellular signal acquisition, is able to amplify low-amplitude action potentials with minimal distortion. The action potentials are assumed to have worst-case amplitudes between 10 microvolts to 150 microvolts and their frequency content is assumed to be between 300 Hz to 6600 Hz. Changing the lower cut-off frequency can be conveniently done by changing the value of one resistor/capacitor. Figure 14 shows the block diagram of the AFE. The complete schematics of the AFE (and other system components) are available in the appendices.

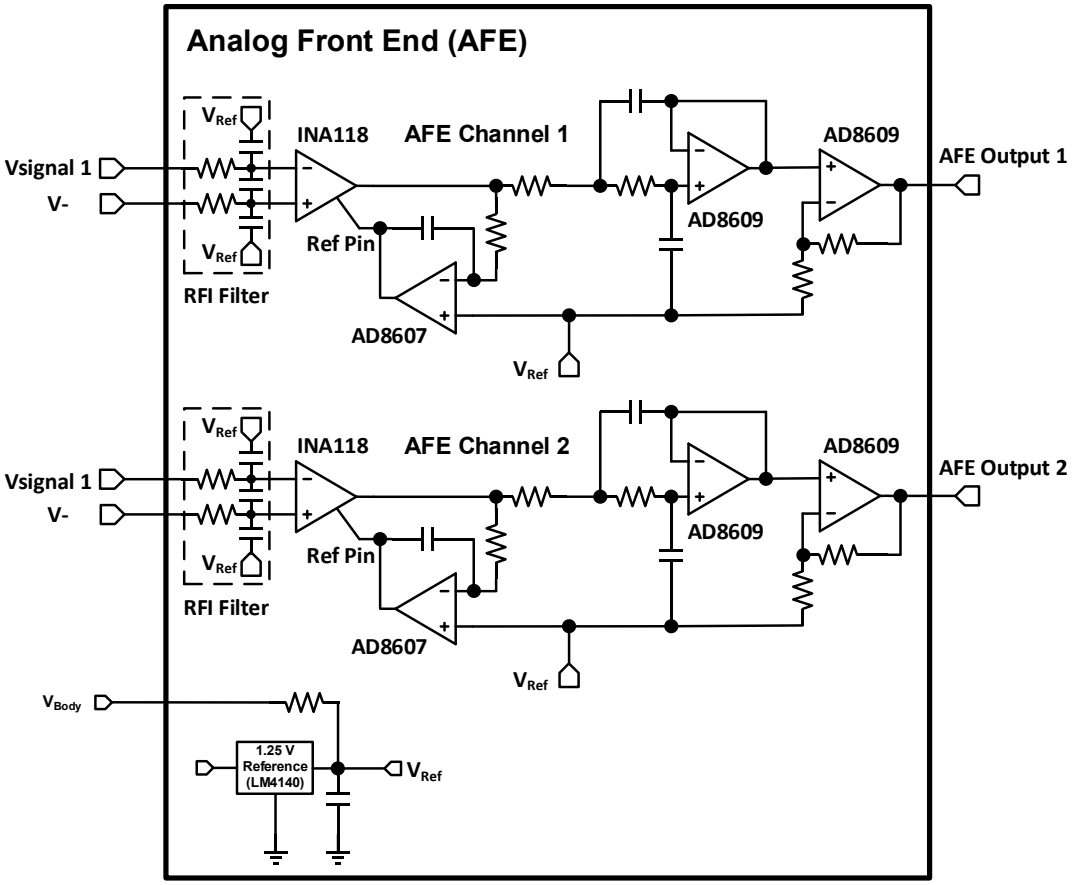


Figure 14. Analog Front End block diagram.

4.4.1 RFI Filter and Preamplifier

The most critical part of the AFE is the preamplifier since it must be as low-noise as possible. In this design, a low-noise low-power instrumentation amplifier from Texas Instruments, INA118 [90], has been used which provides the desired noise characteristics. The INA118 is AC-coupled to the input signals and this is done via another op-amp in a feedback configuration. This feedback adds a high-pass pole to the preamplifier and the cut-off frequency of this pole is determined via the values of one resistor and one capacitor.

This preamplifier is preceded by a passive network, which acts as a radio-frequency interference (RFI) filter [91]. This passive filter is intended to remove the high-frequency signals as the instrumentation amplifiers have very poor CMRR at high frequencies [91].

4.4.2 Mid-Supply Reference

The RFI filter, preamplifier and other parts of the AFE require a stable and low-noise mid-supply reference (a reference voltage equal to almost half the AFE supply voltage). Any noise on this reference voltage will directly affect the low-amplitude signals at the preamplifier terminals. To overcome this issue, a low-noise voltage

reference with high PSRR, LM4140 [92], from Texas Instruments has been used. The mid-supply reference is V_{Ref} in the AFE block diagram. The main reason behind choosing a voltage reference rather than a simple but accurate resistive voltage divider is that the resistive voltage dividers tend to have very poor PSRR characteristics.

4.4.3 Low-Pass Filter

After the pre-amplification, the neural signals are passed through a 2nd-order Sallen-Key filter. This filter and the RFI filter, together, provide the AFE with a low-pass cut-off frequency of almost 6000 Hz.

4.4.4 Second Stage Amplifier

The outputs of the low-pass filters are fed to non-inverting op-amp-based amplifiers where these signals are amplified again to a level that is appropriate for the analog-to-digital converter (ADC).

4.4.5 AFE Power Supply Rails and References

The AFE has a 3.0 volt supply voltage that is provided by the PMU and the reference voltage (generated by the LM4140) is 1.25 volts. The 1.25 volt reference voltage biases the input signals at a point where the preamplifier and other op-amps have the best CMRR. The AFE power supply rail has been chosen to be 3.0 volts so a reliable distance between the battery voltage (~3.6 volts) and the LDO (low-dropout regulator) output voltage is kept. As a result, the LDO PSRR is maximized while the *absolute minimum working voltage* of all AFE parts is respected.

4.5 Optical Stimulation Circuitry

The optical stimulation circuitry is responsible for generating stimulation current waveforms in the stimulation LED terminals. This system consists basically of a current source, based on an op-amp. Figure 15 shows the optical stimulation circuitry block diagram.

The PWM waveform, being the stimulation waveform, is provided by the MCU (microcontroller unit) and it has the same voltage level as the MCU logic outputs i.e., 0 volts and +3.3 volts.

The current source is implemented using an op-amp with negative feedback. The LED anode is directly connected to the battery voltage (~3.6 volts) and its cathode is connected to the current course. The feedback mechanism uses a precision ±1% 0.5 Ω resistor to tune the LED current. Since the resistance of 0.5 Ω is comparable to the resistance of a PCB tracks (even short tracks), in the PCB layout of the headstage, the resistor, the op-amp and Q2 are placed as close as possible; also a ground plane provides a low-impedance ground for all the components of the PCB.

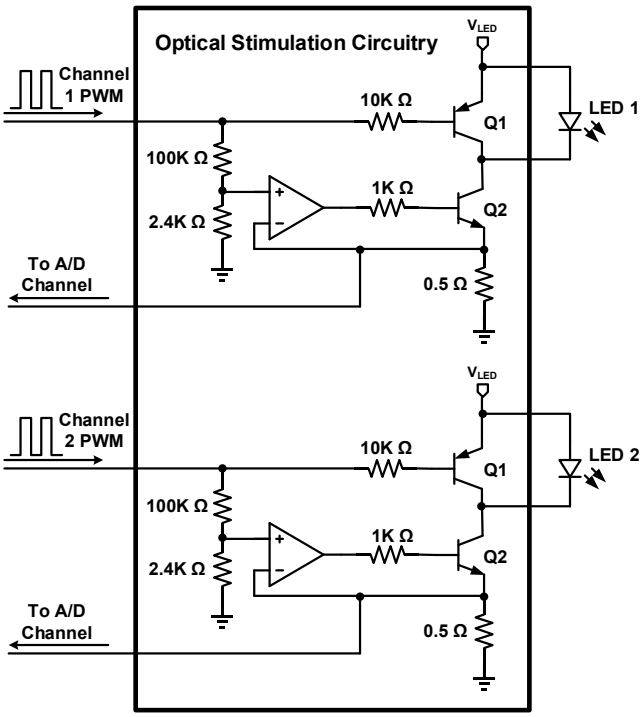


Figure 15. Optical stimulation circuitry block diagram.

In order to control the current and also be informed about the stimulation instances, i.e., when the current is flowing through the LED, the voltage on the 0.5 Ω resistor is monitor by the MCU via one of the A/D (analog-to-digital converter) channels.

4.5.1 LED Current and Sharp Transitions

Since a high amount of current passes though the LED and it needs to be depleted as fast as possible (for the stimulation pattern to have fast rise-/fall-times), Q1 assures that when the PWM is at low state (logical zero), the voltage across the LED is much lower than the conduction voltage (~ 3.3 volts). This allows the LED current waveforms to be sharp i.e., very low rise- and fall-times.

Since the LED current is 150 milliamps when the LED is active, a considerable amount of switching noise can be seen on the battery voltage. This switching noise can be conducted to other components of the headstage. This problem is fully solved by incorporation of high PSRR components and also a power supply filter. More details are available in the PMU section.

4.6 Power Management Unit (PMU)

The PMU is responsible for providing other subsystems of the headstage with stable power supply rails. It consists of two LDOs (Low-Dropout Regulators) with high PSRR and two power supply filters, which remove the high-frequency noise of the power rails. Figure 16 shows the block diagram of the PMU.

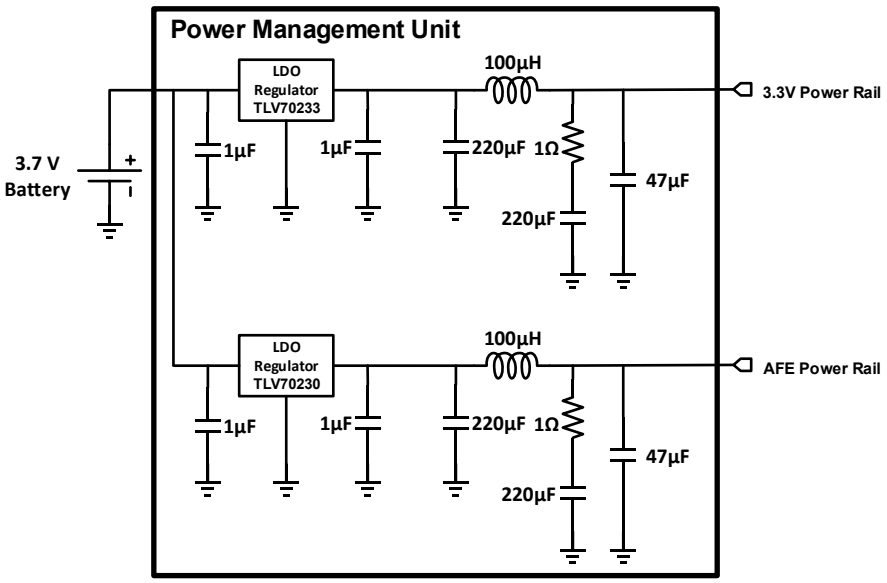


Figure 16. Power management unit block diagram.

The PMU is powered via a 3.7 volt Lithium-Ion rechargeable battery. The battery voltage is high enough so two LDOs can be turned and these two LDOs provide the power for the rest of the components in the headstage system. However, the anodes of the stimulation LEDs are directly connected to the battery's positive terminal rather than any voltage regulator.

It can be seen that one LDO and its power supply filter provides a 3.0 volt power rail for the AFE while the other one provides a 3.3 volt supply for the MCU and the radio transceiver. The reason behind separating the LDOs is 1) minimum noise conduction between the AFE and other components, and 2) a 3.0 volt LDO provides better PSRR than a 3.3 volt LDO when the battery voltage is ~3.6 volts.

4.6.1 Power Supply Filters

The power supply network (Figure 17), being a combination of passive components, provides a very robust removal of high-frequency noise on the LDO outputs. The LDOs that have been chosen (TLV70233 and TLV70230 from Texas Instruments) provide almost 51 dB of PSRR at 100 KHz. The PSRR drops rapidly for higher frequencies. In other words, the PSRR characteristics of these LDOs is very poor at high frequencies. On the other hand, the power supply network is a low-pass network that has a high attenuation at high frequencies. As a result the ensemble PSRR characteristics of the LDOs and the power supply filter provides a high net PSRR at all frequencies. The structure of the power supply filter has been adopted from [93] and its frequency response, after choosing component values, is depicted in Figure 18.

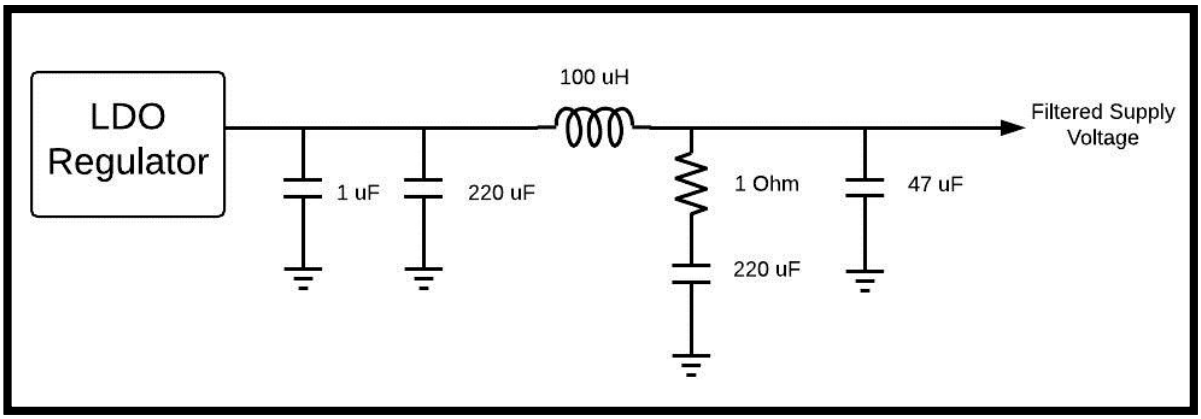


Figure 17. Power supply filter topology.

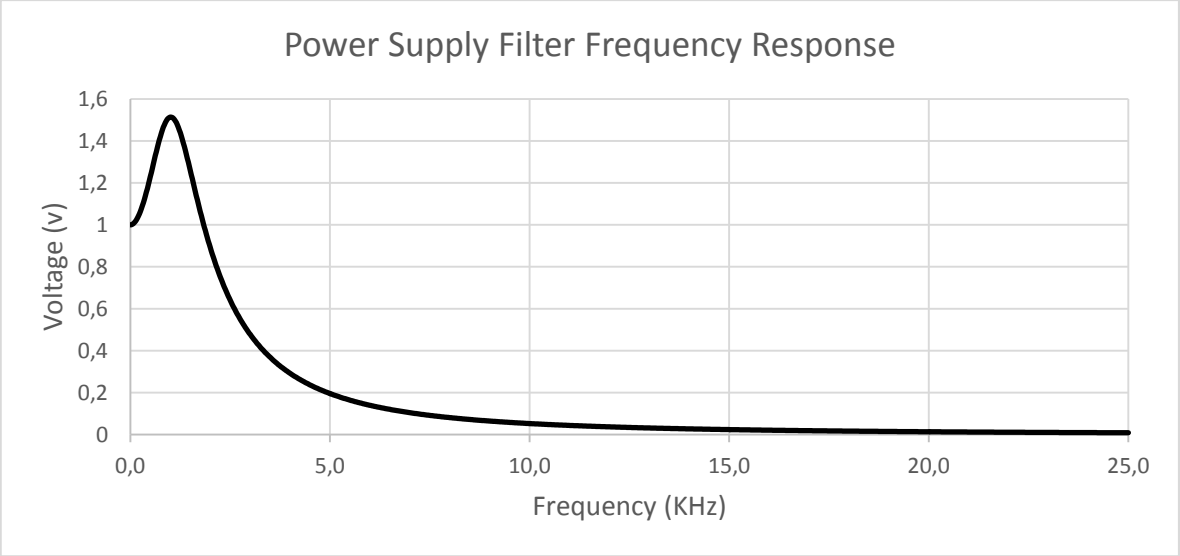


Figure 18. Power supply filter frequency response.

In the headstage system, the noise that the PSRR of LDOs and the power supply filters are targeting to remove is mostly the switching noise of the power supply caused by the LED switching. When incorporating the power supply filters, this noise is significantly reduced and what remains from this noise is taken care of by the high PSRR of the instrumentation amplifiers. This has been proved by measurements and is available in the results in the following chapter.

Finally, it should be mentioned that the power supply filter component values have been chosen for the filter to have the minimum cut-off frequency (the filter has a low-pass behavior). However, lower cut-off frequencies result in bigger electronic components (capacitors, inductors). As a result, there was limitations on the achievable cut-off frequency as the headstage requires small components. The final component values have been chosen as a result of simulations and measurements.

4.7 Microcontroller Unit (MCU)

In this headstage system, a low-power microcontroller, MSP430F5328 from Texas Instruments [94], controls the operation of the headstage. The tasks of this microcontroller include controlling the RF communications, digitizing the output of the AFE and generating the stimulation patterns. This microcontroller, besides occupying a small area on the PCB, has all the necessary peripherals required for the headstage system including the required number of A/D channels and SPI (serial peripheral interface) peripherals. The operating voltage of this microcontroller and its IO (input/output) voltages is 3.3 volts. In order to maximize the power efficiency of this microcontroller an RTOS (real-time operating system) has been used to control the peripherals.

FunkOS [95] has been chosen as the RTOS for its low latency and low memory footprint. Also, FunkOS's programming language, C, its compatibility with MSO430 interrupts and the ease of porting this RTOS to MSP430F5328 have been the driving forces for choosing this RTOS.

During the development phase, the context switching latency of FunkOS has been measured to be almost 35 microseconds, at an operating frequency of 8 MHz, which is the lowest compared to other RTOS like SYS/BIOS, the FreeRTOS and BRTOS. The low context switching latency allows any RTOS task to be interrupted when the A/D buffer is filled. When the A/D buffer is full, the MSP430 DMA (direct memory access) transfers the data to the memory and the microcontroller can send the data to the wireless transceiver.

The firmware running in the MSP430 has the following tasks:

1) One task for receiving commands and configuration from the base station (highest priority).
2) One task for switching the radio transceiver from receiving mode to *idle* mode. This task also synchronizes the beginning of a series of transmission with the base station.
3) One task for transmitting the acquired neural data to the base station when the DMA has finished transferring the data to the pack buffer.
4) One task for putting the microcontroller in sleep mode when no other tasks are being executed.

Figure 19 shows the flowchart of the firmware operation. The complete source code of the firmware is available in the appendices.

When the microcontroller is not executing any task, it is put in sleep mode. In transmitting mode, the microcontroller is awoken every ~800 microseconds, when a 32-byte packet is ready to be transmitted, and the following tasks must be finished before the next packet is ready:

- Context switching (almost 35 microseconds);
- Switching the transceiver mode from Idle to TX (almost 70 microseconds);
- Transferring the 32-byte packet to the transceiver (almost 42 microseconds).

As a result, the MCU can be put to sleep mode for almost 653 microseconds, which equals to 80% of the CPU (central processing unit) time. However, after measuring the sleep time, it has been seen that the MCU is in sleep mode for almost 73% of the time. The reason behind this difference, which translates to less than 7 instructions in MSP430, can be attributed to the overhead added by the execution of some instructions that have not been accounted for.

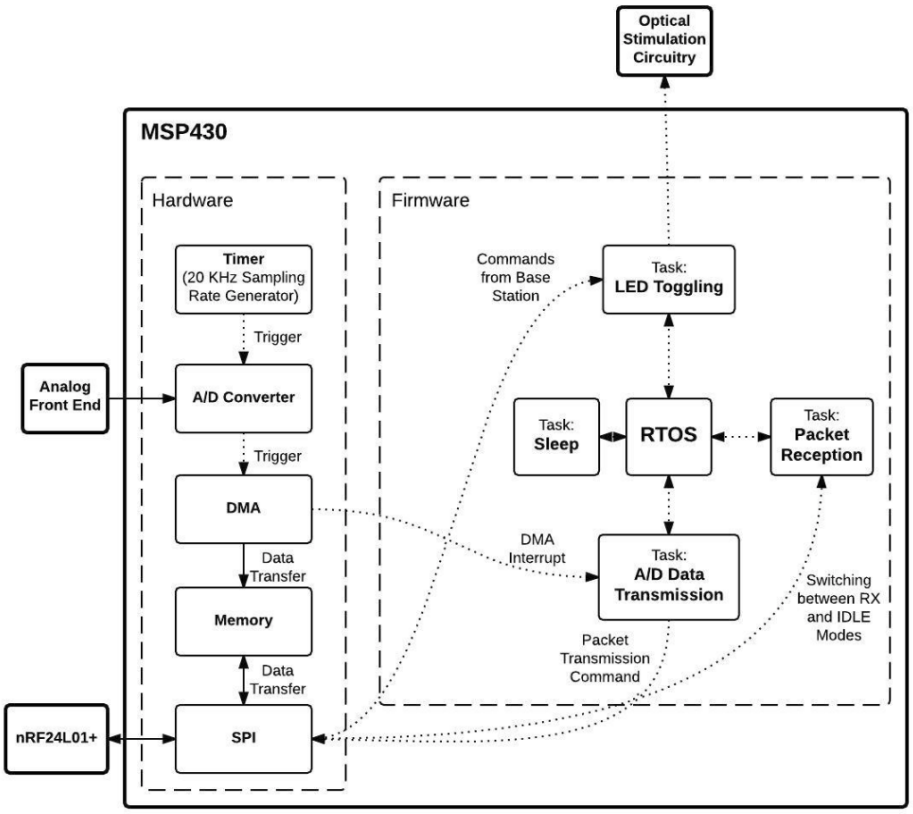


Figure 19. MSP430F5328 firmware flowchart.

4.8 Digital Wireless Transceiver

In order to transmit the received data back to the base station and to receive the stimulation parameters, a modular low-power digital wireless transceiver (nRF24L01+ from Nordic Semiconductor [96]) has been used.

This transceiver is a low-power GFSK radio module operating in the 2.4 GHz ISM (industrial, scientific and medical) band. The raw data of this module can be as high as 2 Mbits per second and the net data rate can be

up to almost 700 kbits per second [96]. An SPI interface is available to control the radio and the chip package is QFN (quad flat no-leads) and measures 4×4 mm^2.

This radio module requires an antenna tuned to the 2.4 GHz band and an antenna matching network consisting of capacitors and inductors. In this project, a chip antenna has been used to decrease the system footprint.

Finally, in order to control the radio module, a C library has been developed that is available in the appendices.

4.9 Electromagnetic Compatibility Considerations

Electromagnetic compatibility (EMC) and electromagnetic interference (EMI) problems can happen in almost all electronic systems except for the most trivial ones. Basically, EMC is about the robustness of a system against incoming noise and interference imposed from the environment while EMI focuses on electromagnetic emissions (radiated and conducted) of a system.

In mixed-signal systems where there are fast high-power high-amplitude digital signals in close proximity of low-amplitude analog signals, close attention must be paid to the EMC and EMI of the system. The headstage that is being discussed in this work is a good example of such mixed signal systems. In this headstage, there are microvolt-scale analog signals, fast digital signals, RF signals and also high-current PWM signals.

4.9.1 EMC/EMI and Self-Interference Problems Associated with the Headstage System

The mentioned digital and high-current signals can have a devastating effect on the low-amplitude analog signals in the following ways:

1) First and most importantly, the high-current LED signals can disturb the battery voltage considerably as the anodes of the LEDs are directly connected to the battery. The imposed (switching) disturbance on the battery voltage can be conducted to the outputs of the LDOs, op-amps and instrumentation amplifiers resulting in very poor signal acquisition in the AFE. This type of disturbance on the AFE power rails can totally distort the signal as the fluctuations on the power rails are much larger than the action potentials. This interference can be grouped as conducted emissions.
2) Since the frequency of the stimulation signals (PWM stimulation pattern) is comparable to the frequency content of the action potentials, if the LED stimulation currents and the action potentials share the same ground plane, the return current of the stimulation pattern will heavily distort the action potentials [97]. This type of interference can also be categorized as conducted emission.

3) Any conductor carrying electrical current can be considered as an antenna [97] and the radiated energy can be picked up by other conductors. The amplitude of the radiated energy depends on the different parameters. However two of them are of crucial importance: the frequency and the amplitude. In the headstage, the digital signals have high-speed harmonics and the LED stimulation currents have high amplitudes. As a result, care must be taken when designing the high PCB to minimize the pickup current on the analog signals [97]. This type of interference can be grouped as radiated emissions.
4) Finally, similar to the return currents of the stimulation currents, the return current path of the digital signals may interfere with the analog signals if the current return paths are the same [97].

Besides the mentioned sources of interference, there might be other sources of interference in the environment with potential to affect the operation of the headstage system.

4.9.2 Solutions to EMC/EMI Problems

In the previous subsection, we described different types of interference and their sources. In this subsection, we will present solutions to these problems. These solutions are all PCB design techniques that can be found in different resources talking about EMC/EMI like [97].

The following techniques can alleviate different problems associated with EMC/EMI [97]:

1) The problem of power rails fluctuations can be solved in mainly two different ways: by using components with high power supply rejection ratio and by using proper bypassing [93].
2) The problem of *shared current return path* can arise in PCBs with power/ground planes as well as PCBs without power planes. One way to solve this problem is to separate the power/ground planes of different system components [97] [98].
3) The problem of radiated emissions can be solved by using ground and/or power planes in the PCB design [97]. When a ground plane is available under high-speed PCB traces, most of the return current passes beneath the track in the opposite direction. As a result, the electromagnetic fields created by these currents will cancel out. This means minimum radiation pickup by other traces. In the case of low-frequency but high-current traces, a return path for the current can be drawn (as a PCB trace) beneath the current-carrying trace.
4) Last but not least, PCBs with ground and power planes tend to be much more robust against environment radiations [97] [98].

4.9.3 Headstage PCB Design

In this work, two PCBs have been design for the headstage system. One PCB is a 4-layer PCB with one ground plane, one power plane and two signal layers. This PCB serves as the system prototype. The other PCB is a 6-layer PCB with the required form factor of the headstage. The 6-layer PCB has 4 signal layers, one power plane and one ground plane.

In both PCB designs, the 4 design techniques as well as other PCB design considerations have been used. In the results chapter, we will present the details of the realized PCBs and their performances.

5 Results and Discussions

5.1 Introduction

In this chapter, we present the results of this project. More specifically, we will present the fabricated devices and their measured performance.

As mentioned in the introduction chapter, the wireless headstages are usually mounted on the animal head in a surgery procedure. In our case, the final headstage consists of two parts: a removable part and a non-removable part. The non-removable part will be mounted on the animal head permanently while the removable part can be placed on or removed from the non-removable part using a board-to-board connector from Molex Inc. [99].

The non-removable part of the headstage is not the goal of this project and is provided by other research groups. It has the appropriate physical shape to be mounted on the animal head and it consists of the LEDs and the electrodes that will be placed in the animal head. The terminals of these LEDs and electrodes will be connected to the board-to-board connector of the non-removable part. The removable part will have a matching connector that will interface to the LEDs/electrodes terminals.

In order to realize the removable part of the headstage system, two PCBs have been designed which can carry the headstage components. One PCB has been designed as a prototype and has larger dimensions than the headstage requirements. The other PCB has the required dimensions of the headstage and is fabricated using the rigid-flex PCB manufacturing technology. The rigid-flex characteristics of the (final) headstage PCB results in low volume and higher robustness to EMC/EMI issues.

Only the prototype PCB has been fabricated and tested at the time of this writing, and the other PCB has been sent for fabrication. All tests that are mentioned in this chapter have been carried out on the prototype PCB. From this point, we will refer to the removable part of the headstage, as *headstage*.

5.2 Headstage PCBs and Their Specifications

In this section, we present the two designed PCBs and their specifications.

5.2.1 Prototype PCB

The fabricated prototype PCB† is a 4-layer PCB having 2 signal layers and 2 power planes. It has all the components that will be mounted on the PCB including the LEDs and the electrodes but its size is intentionally

† This PCB has been mainly designed by Gabriel Gagnon-Turcotte and was edited by Reza Ameli and Alireza Avakh Kisomi.

more than the targeted headstage size so it would be easy to debug. This PCB measures 102×54.2 mm^2 and it has 485 pads, 2573 tracks and 114 vias. The minimum track width/spacing is 0.2 mm and the minimum hole diameter is 0.4 mm. The layer stacking of this PCB is shown in Table 10.

Table 10. Headstage prototype PCB layer stack.

Layer	Type
1	Signal
2	VDD 3.3 v
3	GND
4	Signal

This PCB has been designed larger than the headstage size requirements so it would be easier to solder SMD components and debug. Figure 20 shows the fabricated *prototype* PCB, Figure 21 shows the 2D PCB layout, Figure 22 shows the front view of the 3D PCB layout and Figure 23 shows the back view of the 3D PCB layout.

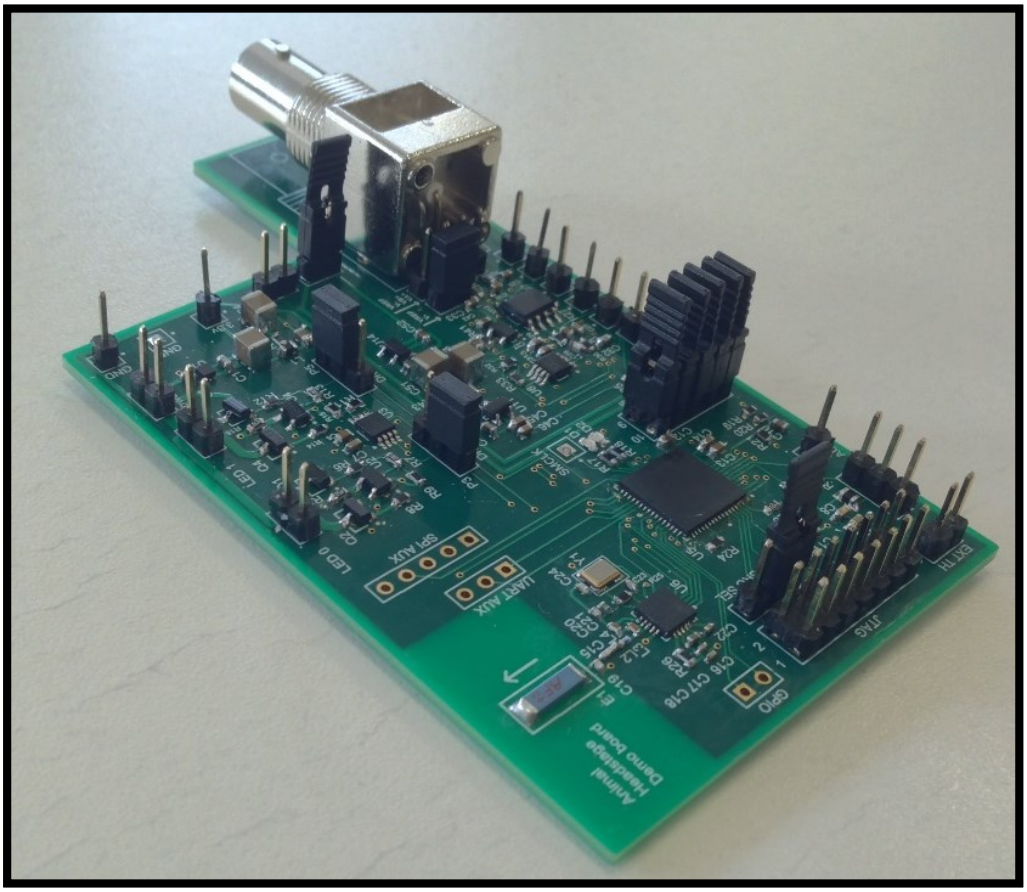


Figure 20. Fabricated headstage prototype PCB.

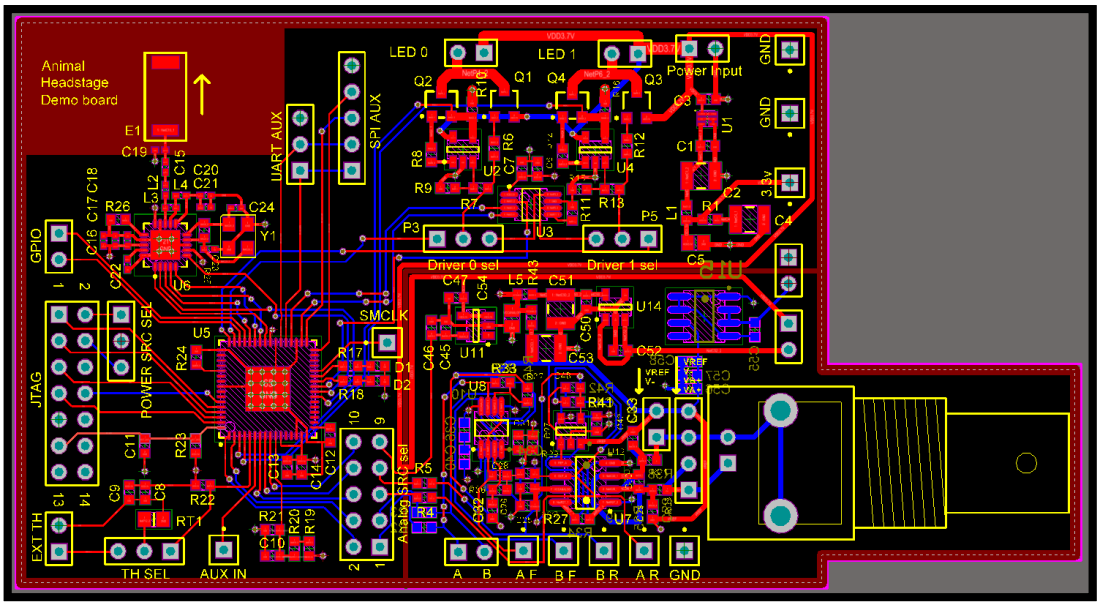


Figure 21. Headstage prototype PCB (2D view).

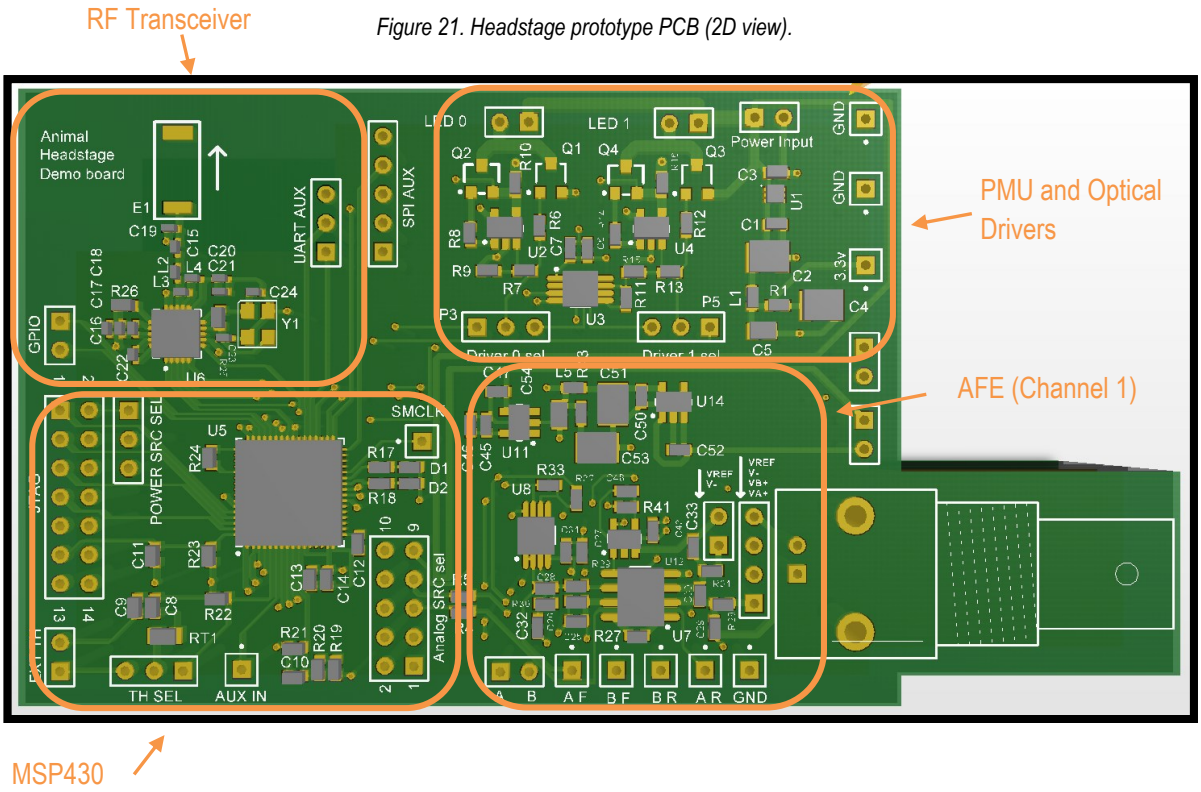


Figure 22. Headstage prototype PCB, front view (3D).

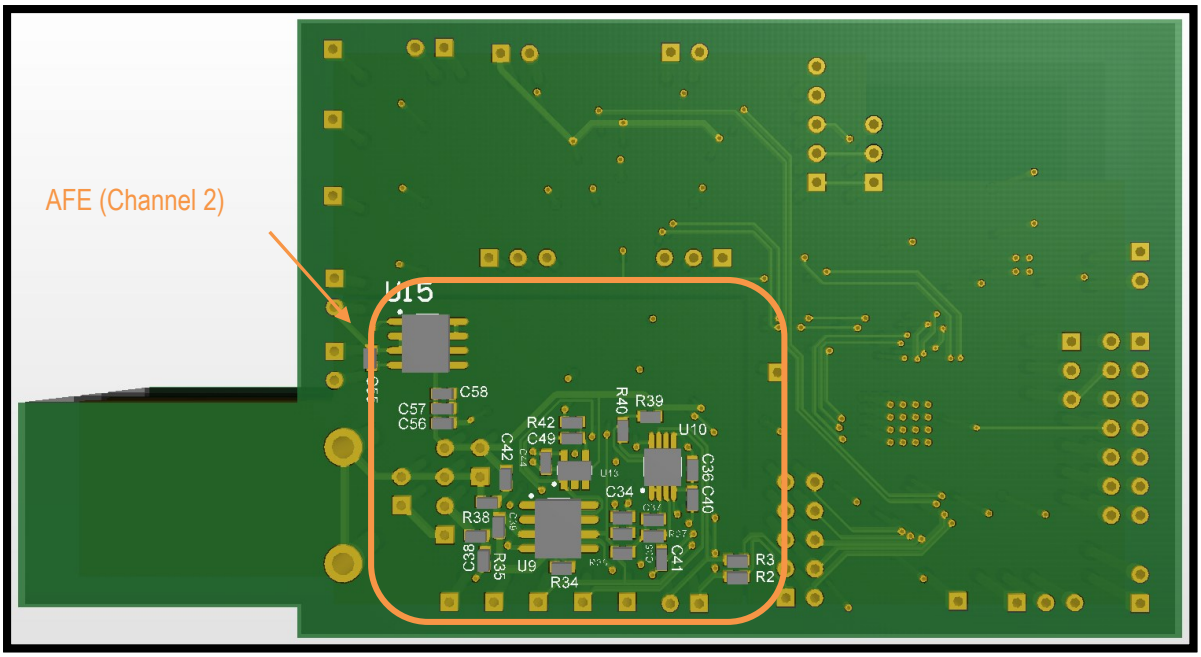


Figure 23. Headstage prototype PCB, back view (3D).

As mentioned before, the prototype PCB has been designed to help with testing and debugging the headstage circuits. It has one BNC connector that can be used to inject signals to the system with minimal added noise. We have tried to use the electronic components that will be used for the final version of the headstage on the prototype as much as possible.

5.2.1.1 EMC/EMI Considerations

In order to ensure the correct functionality of this prototype system, the following EMC/EMI techniques have been used:

1) The power plane and the ground plane are divided into two sections. One section is dedicated to the AFE while the other section is dedicated to all other subsystems of the headstage. This technique ensures that the current return paths of the signals that are not analog will not share the power/ground plane with the AFE. It should be noted that separating power/ground planes means that the plane is divided in two sections that are connected to each other only at a small connection point [97].
2) All digital high-speed traces have ground plane beneath them. This technique assures that the electromagnetic emissions of the high-speed digital signals will be cancelled out by the electromagnetic emissions of their return currents. As a result, the noise that will be picked up by other conductors will be minimum.
3) All traces have power/ground planes. This results into minimum noise picked up by any conductor.

4) Components with high CMRR/PSRR have been used. Also, power supply filters have been incorporated. For more details please refer to Chapter 4.

The prototype PCB has been fully tested and satisfies all the design requirements. All the measured results are presented in the results section in this chapter.

5.2.2 Final Headstage PCB

The final headstage PCB, being the removable part of the headstage system, has been design on a rigid-flex PCB with 6 layers. However this PCB has not been fabricated yet. This PCB will carry the same parts that are mounted on the prototype PCB (expect for the LEDs and the electrodes) but has the required form factor of the final headstage. As mentioned in the introduction of this chapter, the final headstage PCB is fabricated using the rigid-flex technology resulting in less occupied volume and better EMC/EMI robustness.

The final headstage PCB measures 65×20.9 mm^2 when unrolled, has minimum trace width/spacing of 0.2 mm and the minimum hole diameter is 0.2 mm. Table 11 shows the layer stacking of this PCB.

Table 11. Final headstage PCB layer stack.

Layer	Type
1	Signal
2	Power
3	Signal
4	Signal
5	GND
6	Signal

The rationale behind this stacking is that most PCB manufacturers prefer to have signal layers as the inner layers when fabricating rigid-flex PCBs.

This rigid-flex PCB has 3 rigid sections that are connected to each other using the flexible sections. This design strategy eliminates the need for bulky board-to-board connectors. Each rigid section measures 15×21 mm^2 and each flexible section measures 10×21 mm^2.

When the rigid parts are folded together, as is the goal of designing this rigid-flex PCB, the headstages measures less than 20×20 mm^2 depending on the folding angle, as stated by the design requirements. Figure 24 shows the unrolled PCB in 2D view.

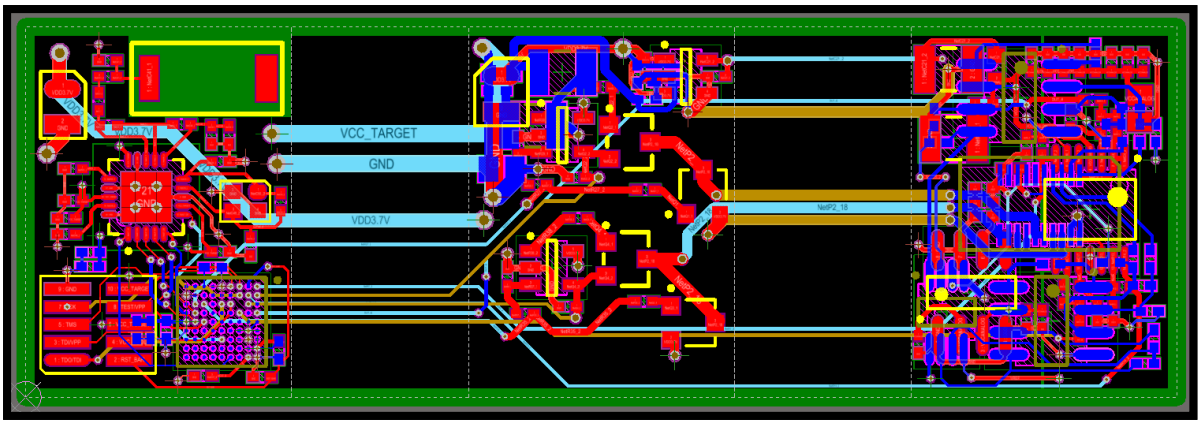


Figure 24. Final headstage PCB unrolled 2D view.

It can be seen that the three rigid parts are connected to each other using light blue and yellow traces. The rigid sections are those populated with components while the flex sections contain only PCB traces. The board-to-board connector is placed on the back of the rigid section at the right side of Figure 24. In Figure 25 and Figure 26, we can see the top view, bottom view and the unrolled final headstage PCB. More specifically, on the right-hand side of Figure 25, the white board-to-board connector can be seen.

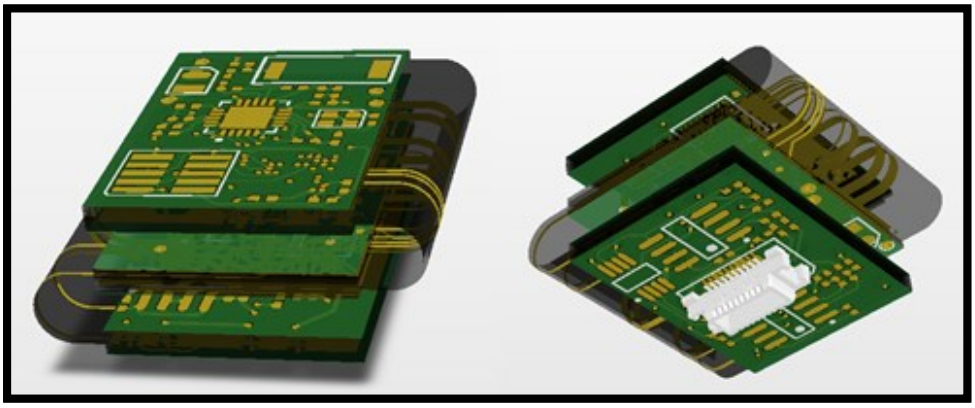


Figure 25. Left: final PCB headstage rolled (top view). Right: final PCB headstage rolled (bottom view).

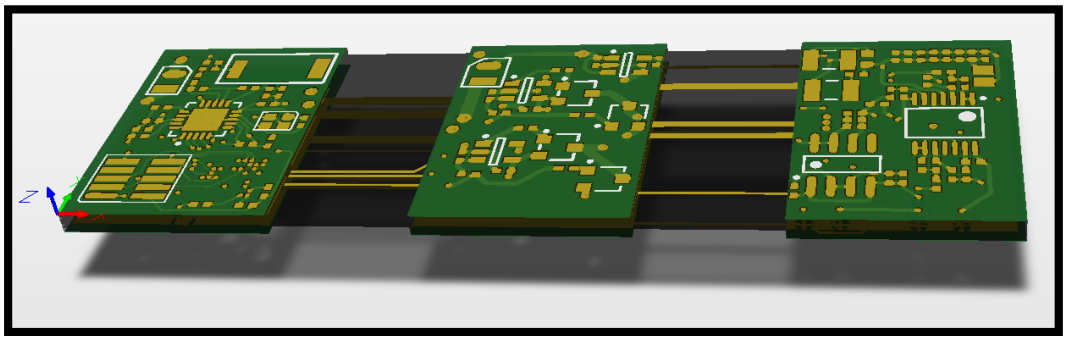


Figure 26. Final PCB headstage unrolled.

While Figure 25 and Figure 26 show the removable part of the headstage system, Figure 27† and Figure 28 show the complete headstage system that will be mounted on the animal head with and without proper packaging. The packaging might be required to protect the headstage from damages caused by animal movements. It can be seen that optical fibers, along with electrodes, will be implemented inside the animal brain.

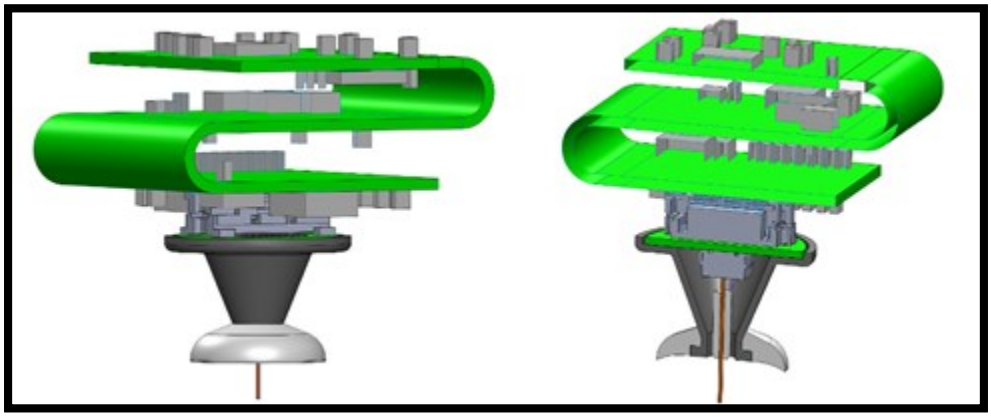


Figure 27. Left: complete headstage system connected to the non-removable part. Right: cross-section view of the complete headstage system.

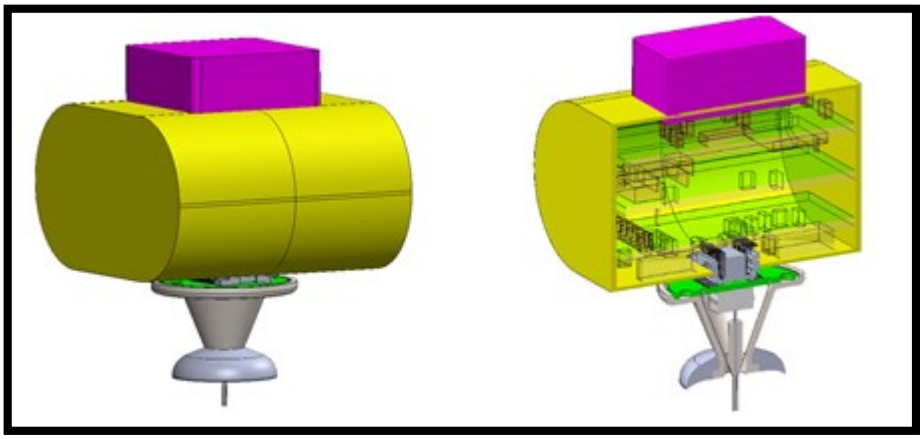


Figure 28. Left: complete headstage system in package. Right: cross-section view of the complete headstage system in package.

5.2.2.1 EMC/EMI Considerations

The EMC/EMI considerations of the final headstage PCB are similar to those of the prototype headstage. However, the final headstage PCB benefits from the inherent separation of different rigid sections. Furthermore, since the rigid section dedicated to the AFE is separate from the other subsystems, more EMC/EMI robustness is resulted.

† Figure 27 and Figure 28 were created by Doric Lenses Inc., Québec, Canada.

5.3 Measured Performance of the AFE

In this section, we present the measured performance of the analog front end (AFE). Three types of test have been carried out to test the AFE. Using these three tests, almost all characteristics of the AFE can be measured.

In the first type of test, the AFE differential inputs have been shorted out together and the noise characteristics of the AFE have been measured. Attention has been paid to the quality of the short circuit as using long wires will result in higher picked up noise. The *input-referred noise* can be calculated by dividing the noise at the AFE output by the overall AFE gain.

In the second type of test, the AFE characteristics including the gain and the cut-off frequencies have been measured using an Agilent 35670A dynamic signal analyzer.

Finally, in the third type of test, synthesized but realistic action potential signals with realistic amplitudes were fed to the AFE and the output of the signal chain (AFE and headstage outputs) were measured.

It should be noted that all the measurements were carried out while the system was working i.e., the LEDs were blinking and the radio was transmitting. In this scenario, all noise sources from other headstage subsystems are present. It can be verified that the high-current LED signals, fast digital signals and the radio RF signals do not affect the performance of the AFE.

Table 12 and Figure 29† provide the measurement results provided by the Agilent dynamic signal analyzer. It can be seen that the input-referred noise of the AFE is low enough for action potentials of 10 microvolts to be detected. Moreover, the AFE characteristics conform to those of the design requirements. The measurement results of the synthetic action potentials will be presented in a later subsection in this chapter.

Table 12. Measured AFE characteristics.

Parameter	Value
Gain	2851 V/V (69.09 dB)
Low Cut-Off Frequency	285 Hz
High Cut-Off Frequency	6580 Hz
Input-Referred Noise	2.1 µV(RMS)
Power Consumption	1 mA @ 3.0 V (3 mW)

† This graph has been prepared by Alireza Avakh Kisomi, M.Sc. student at Laval University, 2015.

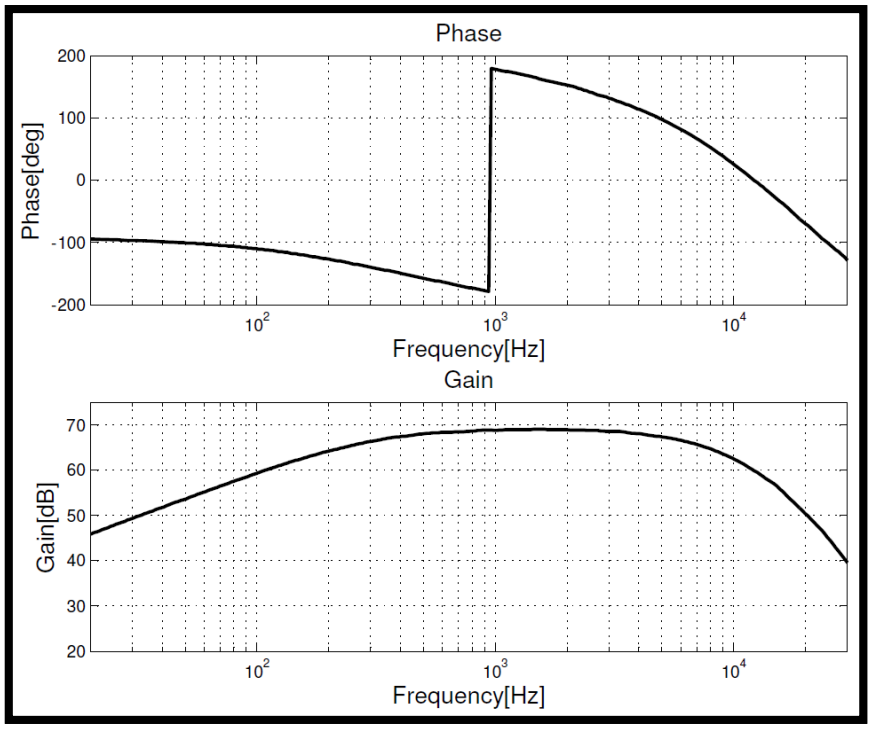


Figure 29. Bode plot of the AFE transfer function.

5.4 Measured Performance of the Optical Stimulation Circuitry

The optical stimulation circuitry has been tested by measuring the voltage on the LED terminals and the current that passes through the LED. The blue LED [87] has been used to carry out the tests. Table 13 summarizes the optical stimulation circuitry performance and Figure 30† shows the voltage on the LED terminals. It can be verified that the rise-time/fall-time of the LED stimulation pattern is negligible compared to the pulse length (i.e., when LED is on) and that the pattern is sharp. This fulfills the needs of the optogenetic experiments having millisecond-scale stimulation patterns.

Table 13. Measured characteristics of the optical stimulation circuitry.

Parameter	Value
LED Terminal Voltage when Active	3.275 V
LED Current when Active	150 mA
Stimulation (PWM) Frequency	1 Hz to 100 Hz
Duty Cycle	10%
Rise Time	1.6 µsec
Fall Time	5.1 µsec
LED input power	491.25 mW

† This screenshot from an oscilloscope has been made by Alireza Avakh Kisomi and Gabriel Gagnon-Turcotte, M.Sc. students at Laval University, 2015.

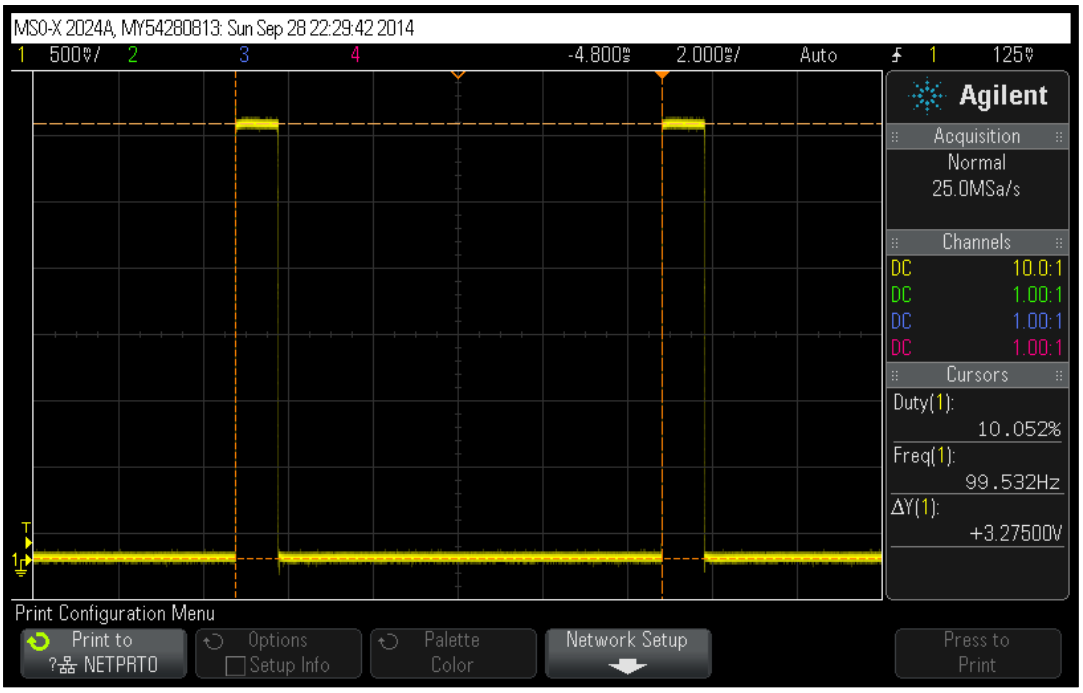


Figure 30. LED voltage during stimulation.

5.5 Power Consumption Measurements

In this section, we present the measured power consumption of all headstage subsystems. The headstage has been powered by a 110 mAh Li-ion battery (GSP061225D2C, Great Power Battery Co., LTD [100]). This battery occupies a volume of 5.7×12×28 mm^3 and has a maximum continuous discharge current of 200 mA. We have chosen this battery as it provides a compromise between size and capacity. Table 14 and Figure 31† provide the power consumption of different headstage subsystems (System LED is a small indication LED used to show the system operation). In our measurements, we have set the stimulation duty cycle to its maximum (10%).

Table 14. Power consumption of headstage subsystems.

Component	Power Consumption When Active
AFE	0.415 mA (1.54 mW)
MCU	4.53 mA (16.8 mW)
Radio Transceiver	4.52 mA (16.7 mW)
Optical Stimulation Circuitry	4.08 mA (15.1 mW)
Stimulation LEDs	15.0 mA (55.5 mW)
System LED	2 mA (7.4 mW)
Total	30.804 mA (113.97 mW)

† This figure has been made by Gabriel Gagnon-Turcotte, M.Sc. student at Laval University, 2015.

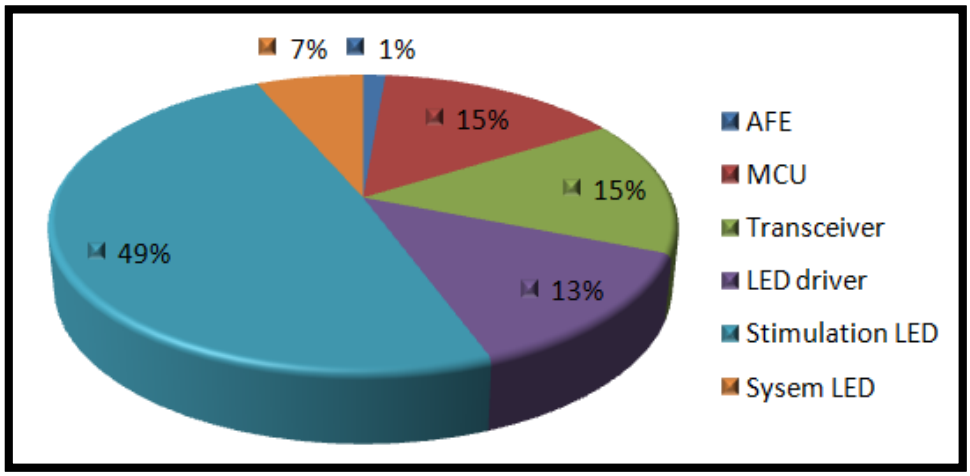


Figure 31. Power consumption of headstage subsystems.

The total average current consumption of the headstage system is 30.804 mA which insures more than 3 hours of continuous lifetime when using a 110 mAh battery as is the case in the headstage system.

5.6 Headstage Outputs with Synthetic Action Potentials as Input

As mentioned in the AFE performance section, one of the tests that have been carried out was feeding the AFE with synthetic but realistic action potentials (these signals have been obtained from the website of authors of [32]). In this section, we will show the outputs of the signal chain when such synthetic signals are fed to the AFE inputs.

Similar to previous tests, this test was carried out when the stimulation LEDs were blinking at their maximum current. In order to realize synthetic action potentials, an arbitrary function generator (Tektronix AFG3101C [101]) has been used. Since the minimum output voltage of this function generator is much larger than the action potential amplitudes, a resistive divider has been used. The resistive divider has been placed in a very close proximity of the AFE input so the picked up noise becomes minimum.

Figure 32† and Figure 33 show the many action potentials acquired by the headstage system. It can be verified that the headstage system acquires signals with high fidelity. However, to test the signal acquisition fidelity in a more quantitative manner, a test has been carried out which will be described later in this section. Figure 34 shows the acquired action potentials and a scaled version stimulation pattern (voltage across the 0.5 ohm resistor in the optical stimulation circuitry) in the same graph. It can be seen that the stimulation pattern, requiring high levels of current has no impact on the quality of the acquired signals.

† Figure 32, Figure 33 and Figure 34 were created by Gabriel Gagnon-Turcotte, M.Sc. student at Laval University, 2015.

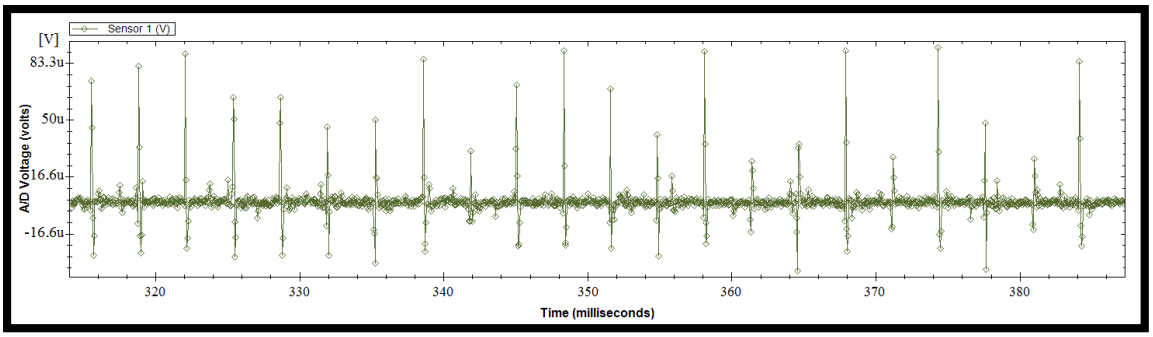


Figure 32. Action potential train acquired by the headstage system.

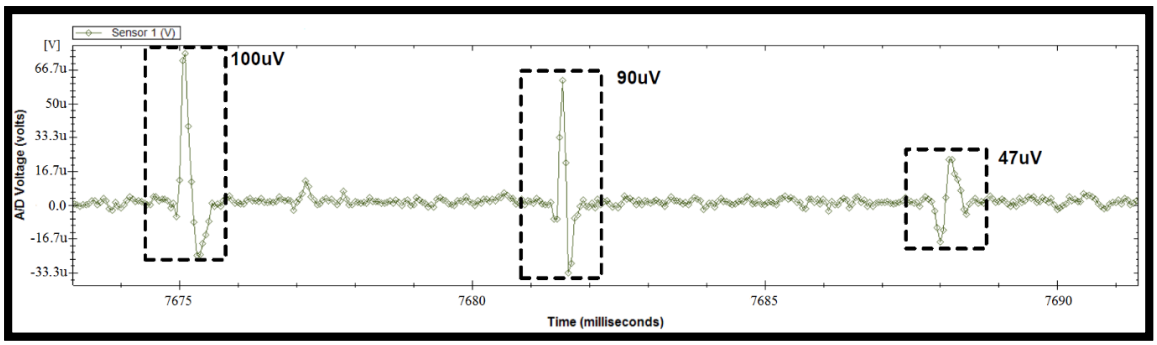


Figure 33. Action potential train acquired by the headstage system (zoomed in).

In order to test the fidelity of the signal acquisition chain, we performed the following test: the 500 action potentials on a 10-second synthetic neural signal were clustered before and after acquisition by the headstage system. We expect that same results yield after the signal is passed through the headstage system. We got the expected results i.e., passing the neural signal through the headstage signal chain has no effect on the shapes of the action potentials. Figure 35† shows the clustered signals.

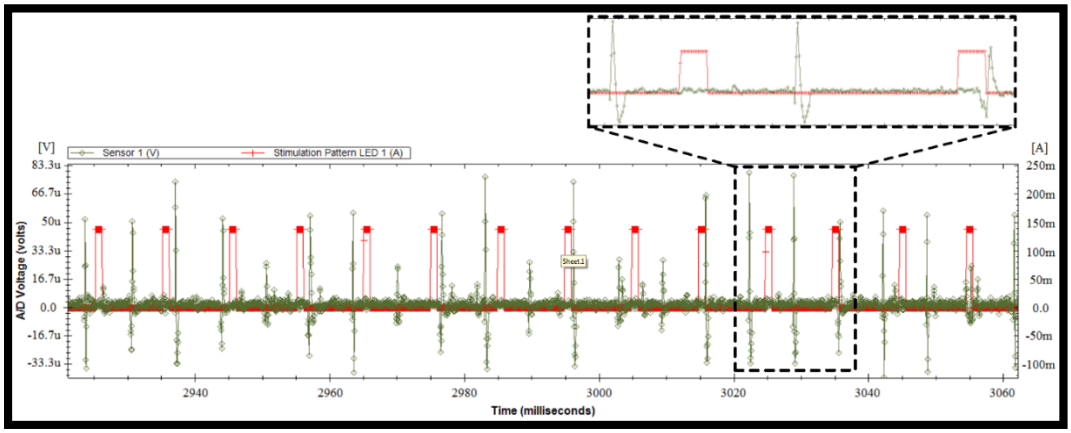


Figure 34. Action potential train acquired by the headstage system along with the stimulation pattern.

† This figure was created by Gabriel Gagnon-Turcotte, M.Sc. student at Laval University, 2015.

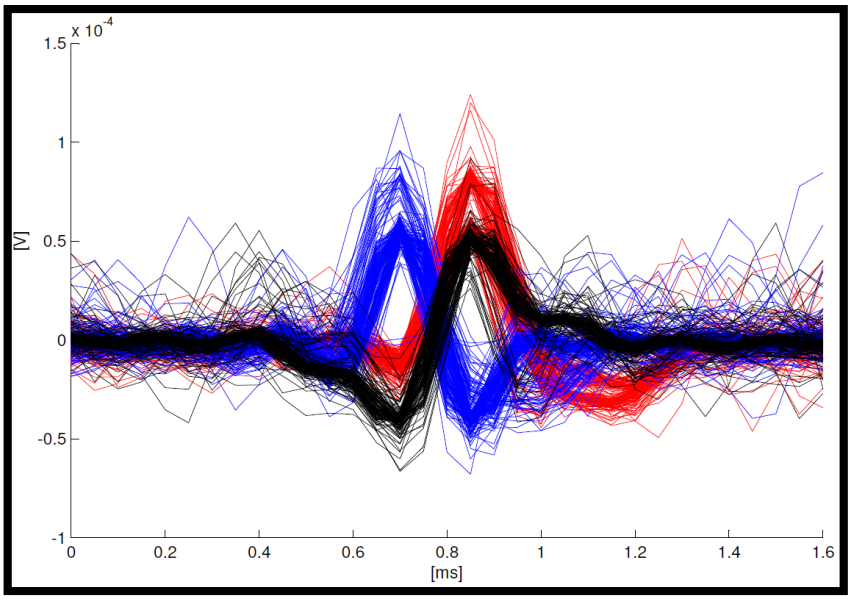


Figure 35. 500 detected, realigned and clustered spikes from a neuronal signal with maximum peak-to-peak voltage of 150 μV.

5.7 Effectiveness of the Power Supply Filter

In the previous sections, we presented the AFE and the complete signal chain outputs. In order to test the effectiveness of the power supply filter discusses in Section 4.6.1, we carried out a test that was exactly the same as the AFE/signal chain tests, but without the power supply filter. To do so, the output of the power supply filter was shorted out to its input so the passive network was ineffective.

Figure 36 shows the fluctuations on battery voltage when the LEDs are blinking at maximum current. It can be seen that there are very sharp and high-voltage (voltage changes of ~80 mV in ~400 nanoseconds) spikes on the battery voltage. These spikes can easily affect the recorded signal as evident in Figure 37.

In Figure 37, it can be seen that the AFE is working normally. However, there are abrupt fluctuations in the output. This proves that, as expected, the power supply filter removes the high-frequency fluctuations of the power supply rails and is vital to the headstage operation as shown in Section 5.6.

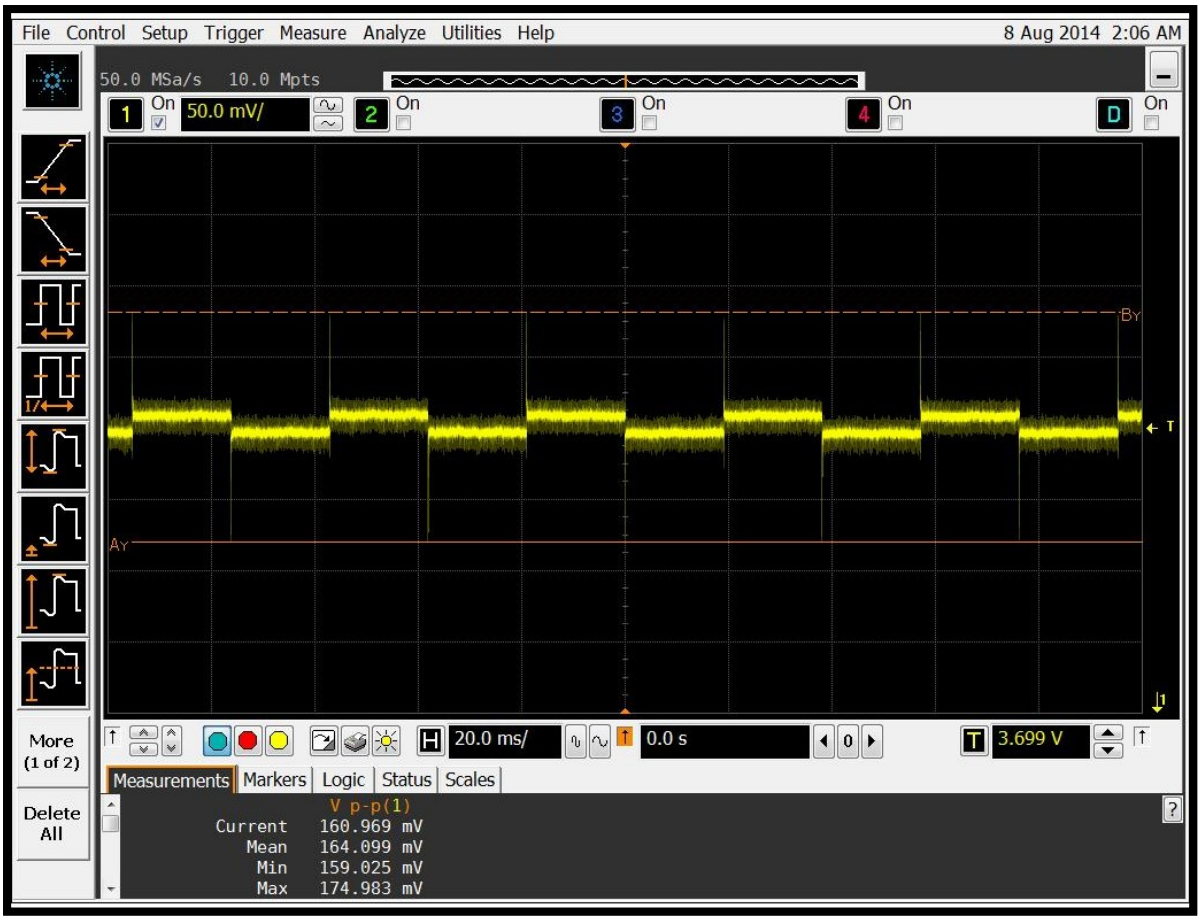


Figure 36. Battery voltage fluctuations when LEDs are blinking at maximum current.

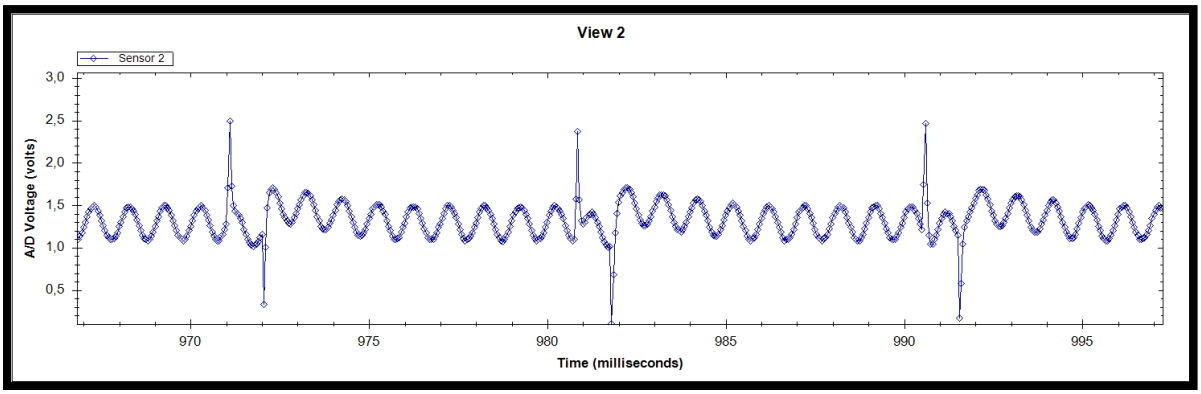


Figure 37. Headstage (signal chain) output without the power supply filter.

5.8 Conclusion

In this chapter, different measured performances of the realized headstage were presented. In this section, we will review the headstage design goals introduced in Chapter 0 and will discuss that these goals have been reached.

5.8.1 Two Optogenetic Stimulation and Recording Channels

In Section 5.4, the voltage waveforms across the LED terminals have been measured and depicted. It can be seen that, when active, the LED voltage is 3.27. This voltage corresponds to a current of 150 mA in the LED [87]. Also, it can be seen in the appendices that there are two LEDs in the designed headstage. Similar to the optical stimulators, the realized headstage has two recording channels and their performance is presented in Section 5.3.

5.8.2 Battery as the Power Source

As evident in sections 4.6 and 5.5, the headstage design is based on a 150 mAh lithium-ion battery. This battery powers the whole system so no power cords are needed.

5.8.3 Optical Stimulation Patterns

In sections 4.5 and 5.4, the design and the measured performance of the optical stimulators were presented, respectively. The high microcontroller clock frequency (compared to the stimulation patterns) and the flexible timer peripherals in the microcontroller result in very accurate control over the generate PWM signals. This is due to the fact that high clock frequency means high time resolution in the microcontroller timers. Figure 30 in Section 5.4 shows the stimulation pattern with the highest frequency. Also, as explained in Section 4.5.1, there is a mechanism available in the headstage that results in very short rise-/fall-times.

5.8.4 Weight and Size Requirements

As discussed in Section 5.2.2, the size of the final headstage rigid-flex PCB (when folded) is less than 20×20×20 mm^3, hence less than the maximum size limit. However, since the final headstage PCB has not been fabricated yet, it is not possible to weight it at this moment. It should be noted that from our previous experience in [17], it is expected that the headstage weight will be satisfying.

Conclusion and Future Works

In this section, we will present the contributions of this work and possible future works that could improve the quality of the wireless optogenetic headstage.

Contributions

The contribution of this work is a novel optogenetic research tool, which allows neuroscientists to perform reliable optogenetic experiments on small rodents. This tool, being a wireless head-mounted device for small animals, has the ability to carry out multi-channel optogenetic stimulation accompanied with simultaneous neural recording. More specifically, this wireless optogenetic headstage is able to stimulate the brain cells using two high-power LEDs and, simultaneously, acquire neural signals from two electrodes samples at 20 kSamples/sec. These features allow researchers to stimulate the brain cell at millisecond scales and investigate the effects of the stimulation in real-time. The optical stimulation patterns created by the high-power LEDs are PWM signals with sharp transients, which in turn adds to the accuracy and usefulness of this device. Also, the analog front end that captures the neural signals has a high robustness against different kinds of noise. Finally, the long life time, the wireless nature, the small size and the low weight of this device allow researchers to carry out long experiments on freely-behaving rodents — a field of study that requires more and more accurate devices to carry more and more sensitive experiments. According to a comprehensive literature review at the time of the writing this thesis, and to the author's knowledge, such optogenetic research tool does not exist in the industry or in the scientific community.

The fact that this headstage fulfills all the requirements is due to experience that was acquired by the author while working on the first version of this wireless headstage. The goal of the mentioned headstage was to have a wireless device that would not need any wires or batteries to operate and would have two electrophysiological recording channels accompanied by one optogenetic stimulation channel. The mentioned optogenetic headstage was powered by an inductive wireless power-delivery link. Many subtle challenges were faced while designing this device and the acquired experience allowed the author to work on the second headstage with confidence.

During the course of this project, the following papers have been published:

1) **R. Ameli, A. Mirbozorgi, J.-L. Néron, Y. LeChasseur and B. Gosselin, "A Wireless and Batteryless Neural Headstage with Optical," in Engineering in Medicine and Biology Society (EMBC), Osaka, 2013**: This paper presents the first version of the wireless optotgenetic heastage based on which the second headstage was designed. The first version has two recording channels but only one optical stimulation channel and occupies a larger space than the second version. The

power source of this headstage is a wiresless power delivery link which can in some scenarios limit the movability of the animal test subjects.

2) **S. Mirbozorgi, R. Ameli, M. Sawan and B. Gosselin, "Towards a wireless optical stimulation system for long term in-vivo experiments," in Engineering in Medicine and Biology Society (EMBC), Chicago, 2014**: This paper presents the design of a wireless power delivery chamber where a small rodent can move with a headstage on its head. The headstage can receive its power from the magnetic field generated by the chamber. A power delivery chamber similar to this chamber has been used in the development of the first version of wireless batteryless optogenetic headstage.

3) **G. Gagnon-Turcotte, C.-O. Dufresne Camaro, A. Avakh, R. Ameli and B. Gosselin, "A Wireless Multichannel Optogenetic Headstage with on-the-Fly Spike Detection," in International Symposium on Circuits and Systems, Lisbon, 2015**: This paper presents the implementation of a light-weigth spike detection algorithm on the wireless headstage that has been developed during this project. The spike detection mechanism is based on the absolute value operator (see section 3.3.2) and it is shown that using real-time spike detection results in reduced power consumption in the wireless transmitter.

4) **G. Gagnon-Turcotte, A. Avakh Kisomi, R. Ameli, C.-O. Dufresne Camaro, Y. LeChasseur, J.-L. Neron, P. Brule Bareil, P. Fortier, C. Bories, Y. De Koninck, and B. Gosselin, "A Wireless Optogenetic Headstage with Multichannel Electrophysiological Recording Capability", Sensors, 2015 (Pending review)**: This paper presents the research work that has been done in this thesis from the point of view of an electronic designer that aims to design a wireless optogenetic headstage based on COTS components. Desirable characteristics of optogenetic wireless headstages as well as different issues that might be encountered during the design phase and their solutions are also presented in this paper.

Challenges

Similar to the first version of this wireless headstage, many design challenges were face during different design and fabrication phases of this project. The realized wireless headstage consists of analog, digital, RF and mixed-signal subsystems with different requirements that must work simultaneously together. The mixed-signal nature of the headstage, its small form factor and the small power source powering the system, made this project challenging in a multi-disciplinary manner.

The optical stimulation circuitry and the battery had to provide the stimulation LEDs with exactly 150 mA of current. This requirement necessitated LED terminal voltages very close to the battery voltage, which in turn

required a feedback network with very low resistor values — close to the resistance of the PCB tracks. Also the switching behavior of the LEDs resulted in high-amplitude conducted high-frequency and low-frequency noise waveforms on the power rails of the other subsystems.

The problem of switching noise on the power rails manifested itself mostly in the operation of the analog front end — an amplifier with very strict noise characteristics. In order to realize the analog front end with the required noise characteristics, all noise sources must have been identified and taken care of. Four types of noise were present in the analog front end circuitry: 1) the inherent noise of the first-stage amplifier, 2) the common-mode noise at the inputs of the first-stage amplifier, 3) the low-frequency noise on the AFE power rails and 4) the high-frequency noise of the AFE power rails.

The first and second noise sources, pertaining to the first-stage amplifier, were solved by using a low-noise instrumentation amplifier with high CMRR. The low-frequency noise present on the AFE power rails was partially reduced by a passive power supply filter and partially by the high PSRR of the instrumentation and operation amplifiers. Finally, the high-frequency noise of the AFE power rails was filtered out by the power supply filter. It should also be mentioned that the PCB design techniques used in the design of the headstage were also effective in removing different types of noise. Besides the efforts that have been made in the AFE to reduce noise, the use of low noise high-PSRR linear regulators in the power management unit helped with noise problems too. The power management unit consists of two linear regulators with different outputs for different subsystems.

Alongside with the hardware design of the headstage, the firmware design turned out to be challenging too. More specifically, the use of a RTOS to increase the manageability of the code resulted in less CPU time for the useful code. At the same time, the requirements of low power consumption meant lower CPU clock rates so the firmware designer had to make sure that, with the added delays of the RTOS, the code would work perfectly in real-time without losing any packets or data samples.

Last but not least, the PCB design of the final headstage required very careful component placement and even floor planning ahead of the placement process as the PCB size had to be small in comparison with a rodent body. Also, the small form factor of the PCB and the EMC/EMI issues that would exist without careful PCB design made the design challenging above the challenges of the circuit design. To address all the EMC/ECI issues, a six-layer flex-rigid PCB technology was adopted and allowed us to find a solution to all design problems.

The headstage system, after design and fabrication, went through different types of test that covered the functionality of the AFE, the optical stimulation circuitry, the MPU and the radio transceiver. In these tests the

correctness of the functionality of each subsystem was tested while all other subsystems were working. In order to emulate the real-life signal acquisition, realistic action potential waveforms at realistic microvolt-scale amplitudes were synthesized and fed to the system. The designed headstage passed all the tests.

Future Works

Similar to all other efforts in the field of engineering and science, this work can be improved. The foreseen improvements can be mainly categorized as 1) animal tests, 2) improvement in the hardware and 3) adding signal processing capability to the headstage.

Animal tests: all the tests, carried out on this research tool, have been done through emulated neural recording scenarios i.e., a signal generator with realistic outputs has been used instead of placing the electrodes in an animal brain. Carrying out tests on real animals results in more insight into the quality of signal acquisition. The animal tests should be carried out using the prototype system as well as the final headstage system.

Improvements in the hardware: in this project, the hardware of all subsystems was designed in a way that would fulfill all the design requirements perfectly. This practice might have resulted in overengineering in some areas. The subsystem that is most likely subject to such overengineering is probably the analog front end as relatively expensive and low-noise components were used. The design of the AFE can probably be less expensive/robust while still fulfilling all the requirements. Also in the PMU, it might be possible to use less expensive linear regulators that have inferior noise characteristics but are still fully functional in the headstage.

Adding signal processing capability to the headstage: Chapter 0 discussed different neural signal processing algorithms suitable for neural recording embedded systems. The possibility of implementing these algorithms in the realized system must be investigated in the current headstage. It is expected that by increasing the clock frequency of the MPU, low-overhead spike detection algorithms can be implemented on this system. Besides, by using more powerful processors and/or FPGAs, many different signal processing algorithms including spike detection, spike compression and spike sorting can be implemented in real-time in the headstage. Design and implementation of such algorithms require careful fixed-point simulation using banks of neural signals. An example of implementation of such algorithms is presented in [83].

References

[1] H. Blumenfeld, Neuroanatomy Through Clinical Cases, Sinauer Associates Inc, 2010.

[2] B. J. Sadock and V. A. Sadock, Kaplan and Sadock's Synopsis of Psychiatry: Behavioral Sciences/Clinical Psychiatry, 10 ed., Lippincott Williams & Wilkins, 2007.

[3] B. Gosselin, A. Ayoub, J.-F. Roy, M. Sawan, F. Lepore, A. Chaudhuri and D. Guitton, "A Mixed-Signal Multichip Neural Recording Interface With Bandwidth Reduction," *IEEE Transactions on Biomedical Circuits and Systems,* vol. 3, pp. 129,141, 2009.

[4] B. Gosselin, "Recent Advances in Neural Recording Microsystems," *Sensors,* vol. 11, no. 5, pp. 4572-4597, 2011.

[5] A. Sharma, L. Rieth, P. Tathireddy, R. Harrison and F. Solzbacher, "Long term in vitro stability of fully integrated wireless neural interfaces based on Utah slant electrode array," *Applied Physics Letters,* vol. 96, no. 7, pp. 073702,073702-3, 2010.

[6] S. D. Coates, Neural Interfacing: Neural Interfacing: Forging the Human-Machine Connection (Synthesis Lectures on Biomedical Engineering), 1st ed., Morgan and Claypool Publishers, 2008.

[7] "Cochlear Implants," [Online]. Available: http://www.canadianaudiology.ca/consumer/cochlear-implants.html. [Accessed 07 10 2014].

[8] "Visual Prosthesis (Retinal Implant & Biohybrid Retinal Implant)," [Online]. Available: http://www.io.mei.titech.ac.jp/research/retina/. [Accessed 07 10 2014].

[9] "The Boston Retinal Implant Project," [Online]. Available: http://www.bostonretinalimplant.org/. [Accessed 07 10 204].

[10] R. Kwok, "Neuroprosthetics: Once more, with feeling," *Nature,* vol. 497, no. 7448, 2013.

[11] L. R. Hochberg, D. Bacher, B. Jarosiewicz, N. Y. Masse, J. D. Simeral, J. Vogel, S. Haddadin, J. Liu, S. S. Cash, P. v. d. Smagt and J. P. Donoghue, "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm," *Nature,* vol. 485, no. 7398, p. 372–375, 2012.

[12] A. K. Senapati, P. J. Huntington, S. C. LaGraize, H. D. Wilson, P. N. Fuchs and Y. B. Peng, "Electrical stimulation of the primary somatosensory cortex inhibits spinal," *Brain Research,* vol. 1057, no. 1-2, p. 134–140, 2005.

[13] A. R. Houweling and M. Brech, "Behavioural report of single neuron stimulation in somatosensory cortex," *Nature,* vol. 451, pp. 65-68, 2008.

[14] M. Halpern and J. Fallon, "Current Waveforms for Neural Stimulation-Charge Delivery With Reduced Maximum Electrode Voltage," *IEEE Transactions on Biomedical Engineering,* vol. 57, no. 9, pp. 2304,2312, 2010.

[15] E. S. Boyden, F. Zhang, E. Bamberg, G. Nagel and K. Deisseroth, "Millisecond-timescale, genetically targeted optical control of neural activity," *Nat. Neurosci.,* pp. 1263 - 1268, 2005.

[16] P. Degenaar, B. McGovern, R. Berlinguer-Palmini, N. Vysokov, N. Grossman, V. Pohrer, E. Drakakis and M. Neil, "Individually addressable optoelectronic arrays for optogenetic neural stimulation," in *Biomedical Circuits and Systems Conference (BioCAS)*, Paphos, 2010.

[17] R. Ameli, A. Mirbozorgi, J.-L. Néron, Y. LeChasseur and B. Gosselin, "A Wireless and Batteryless Neural

Headstage with Optical," in *35th annual Conf. of EMBC*, 2013, pp. 5662-5665.

[18] D. Huber, L. Petreanu, N. Ghitani, S. Ranade, T. Hromadka and S. K. MainenZach, "Sparse optical microstimulation in barrel cortex drives learned behaviour in freely moving mice," *Nature Letters,* vol. 451, pp. 61-64, 2008.

[19] H. E. Covington, M. K. Lobo, I. Maze, V. Vialou, H. J. M., S. Zaman, Q. LaPlant, E. Mouzon, S. Ghose, A. C. Tamminga, R. L. Neve, K. Deisseroth and E. J. Nestler, "Antidepressant Effect of Optogenetic Stimulation of the Medial Prefrontal Cortex," *Journal of Neuroscience,* vol. 30, no. 48, p. 16082–16090, 2010.

[20] V. Gradinaru, M. Mogri, K. R. Thompson, J. M. Henderson and K. Deisseroth, "Optical deconstruction of parkinsonian neural circuitry," *Science,* vol. 324, no. 5925, pp. 354-359, 2009.

[21] "Optogenetics and Behavior," [Online]. Available: http://www.openoptogenetics.org/index.php?title=Journal_Articles#Optogenetics_and_Behavior.

[22] "Optogenetics: Method of the Year 2010," *Nature Methods,* 2010.

[23] A. Fiala, A. Suska and O. M. Schlüter, "Optogenetic approaches in neuroscience," *Current Biology,* vol. 20, no. 20, p. R897–R903, 2010.

[24] S. K. Arfin, M. A. Long, M. S. Fee and R. Sarpeshkar, "Wireless Neural Stimulation in Freely Behaving Small Animals," *J. Neurophysiol.,* 2009.

[25] B. Wanga, J. Zhouc, P. Carneya and H. Jiang, "A novel detachable head-mounted device for simultaneous EEG and photoacoustic monitoring of epilepsy in freely moving rats," *Neuroscience Research,* 2014.

[26] Q. Tang, M. Brecht and A. Burgalossi, "Juxtacellular recording and morphological identification of single neurons in freely moving rats," *Nature Protocols,* vol. 9, no. 10, p. 2369–2381, 2014.

[27] M. S. Chae, Z. Yang, M. R. Yuce, L. Hoang and W. Liu, "A 128-Channel 6 mW Wireless Neural Recording IC With Spike Feature Extraction and UWB Transmitter," *IEEE Transactions on Neural Systems & Rehabilitation Engineering,* vol. 17, no. 4, pp. 312-321, 2009.

[28] B. C. George, "The Neuron," [Online]. Available: http://webspace.ship.edu/cgboer/theneuron.html.

[29] T. H. Bullock and G. A. Horridge, Structure and Function in the Nervous Systems of Invertebrates, 1965.

[30] "Nervous Energy," [Online]. Available: http://www.helcohi.com/sse/body/nervous.html.

[31] "Lights, Camera, Action Potential," [Online]. Available: https://faculty.washington.edu/chudler/ap.html.

[32] J. Martinez, C. Pedreira, M. J. Ison and R. Quian Quiroga, "Realistic simulation of extracellular recordings," *Journal of Neuroscience Methods,* vol. 184, no. 2, p. 285–293, 2009.

[33] R. Quian Quiroga, Z. Nadasdy and Y. Ben-Shaul, "Unsupervised Spike Detection and Sorting with Wavelets and Superparamagnetic Clustering," *Neural Computation,* vol. 16, no. 8, pp. 1661-1687, 2004.

[34] S. Shahid, J. Walker and L. S. Smith, "A New Spike Detection Algorithm for Extracellular Neural Recordings," *IEEE Transactions on Biomedical Engineering,* vol. 57, no. 4, pp. 853-866, 2010.

[35] D. A. Henze, Z. Borhegyi, J. Csicsvari, A. Mamiya, K. D. Harris and G. Buzsáki, "Intracellular Features Predicted by Extracellular Recordings in the Hippocampus In Vivo," *Journal of Neurophysiology,* vol. 84, no. 1, pp. 390-400, 2000.

[36] S. Gibson, J. W. Judy and D. Markovic, "Technology-Aware Algorithm Design for Neural Spike Detection, Feature Extraction, and Dimensionality Reduction," *IEEE Transactions on Neural Systems &*

Rehabilitation Engineering, vol. 18, no. 5, pp. 469-478, 2010.

[37] L. A. Camuñas-Mesa and R. Q. Quiroga, "A detailed and fast model of extracellular recordings," *Neural Computation,* vol. 25, no. 5, pp. 1191-1212, 2013.

[38] P. T. Thorbergsson, M. Garwicz, J. Schouenborg and A. J. Johansson, "Computationally efficient simulation of extracellular recordings with multielectrode arrays," *Computational Neuroscience,* vol. 211, no. 1, p. 133–144, 2012.

[39] V. Karkare, S. Gibson and D. Markovic, "A 130-W, 64-Channel Neural Spike-Sorting DSP Chip," *IEEE Journal of Solid-State Circuits,* vol. 46, no. 5, pp. 1214-1222, 2011.

[40] B. Yu, T. Mak, X. Li, F. Xia, A. Yakovlev, Y. Sun and C.-S. Poon, "Real-Time FPGA-Based Multichannel Spike Sorting Using Hebbian Eigenfilters," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 1, no. 4, pp. 502-515, 2011.

[41] R. Harrison, P. Watkins, R. Kier, R. Lovejoy, D. Black, B. Greger and F. Solzbacher, "A Low-Power Integrated Circuit for a Wireless 100-Electrode Neural Recording System," *Solid-State Circuits, IEEE Journal of,* vol. 42, no. 1, pp. 123-133, 2007.

[42] R. Chandra and L. M. Optican, "Detection, classification, and superposition resolution of action potentials in multiunit single-channel recordings by an on-line real-time neural network," *IEEE Transactions on Biomedical Engineering,* vol. 44, no. 5, p. 403–412, 1997.

[43] V. J. Prochazka and H. H. Kornhuber, "On-line multi-unit sorting with resolution of superposition potentials," *Electroencephalography and Clinical Neurophysiology,* vol. 34, no. 1, p. 91–93, 1973.

[44] Y. Perelman and R. Ginosar, "An Integrated System for Multichannel Neuronal Recording With Spike/LFP Separation, Integrated A/D Conversion and Threshold Detection," *Biomedical Engineering, IEEE Transactions on,* vol. 54, no. 1, pp. 130-137, 2007.

[45] B. Gosselin and M. Sawan, "An Ultra Low-Power CMOS Automatic Action Potential Detector," *IEEE Transactions on Neural Systems & Rehabilitation Engineering,* vol. 17, no. 4, pp. 246-353, 2009.

[46] H. Gao, R. M. Walker, P. Nuyujukian, K. A. Makinwa, K. V. Shenoy, B. Murmann and T. H. Meng, "HermesE: A 96-Channel Full Data Rate Direct Neural Interface in 0.13 mCMOS," *IEEE Journal of Solid-State Circuits,* vol. 47, no. 4, pp. 1043-1055, 2012.

[47] K. H. Kim and S. J. Kim, "A Wavelet-Based Method for Action Potential Detection From Extracellular Neural Signal Recording With Low Signal-to-Noise Ratio," *IEEE Transactions on Biomedical Engineering,* vol. 50, no. 8, pp. 999-1011, 2003.

[48] R. R. Harrison, R. J. Kier, C. A. Chestek, V. Gilja, P. Nuyujukian, S. Ryu, B. Greger, F. Solzbacher and K. V. Shenoy, "Wireless Neural Recording With Single Low-Power Integrated Circuit," *IEEE Transactions on Neural Systems & Rehabilitation Engineering,* vol. 17, no. 4, pp. 322-329, 2009.

[49] R. H. Olsson and K. D. Wise, "A Three-Dimensional Neural Recording Microsystem With Implantable Data Compression Circuitry," *IEEE Journal of Solid-State Circuits,* vol. 40, no. 12, pp. 2796-2804, 2005.

[50] P. Hegemann and S. Sigrist, Optogenetics (Dahlem Workshop Reports), Walter de Gruyter, 2013.

[51] "What is Optogenetics?," [Online]. Available: http://optogenetics.weebly.com/what-is-it.html.

[52] S. Zhao, C. Cunha, F. Zhang, Q. Liu, B. Gloss, K. Deisseroth, G. J. Augustine and G. Feng, "Improved expression of halorhodopsin for light-induced silencing of neuronal activity," *Brain Cell Biol,* vol. 36, no. 1-4, pp. 141-154, 2008.

[53] A. M. Sodagar, G. E. Perlin, Y. Yao, K. Najafi and K. Wise, "An Implantable 64-Channel Wireless

Microsystem for Single-Unit Neural Recording," *Solid-State Circuits, IEEE Journal of,* vol. 44, no. 9, pp. 2591-2604, 2009.

[54] D. S. Schregardus, A. W. Pieneman, A. Ter Maat, R. F. Jansen, B. J. F. and M. L. Gahr, "A lightweight telemetry system for recording neuronal activity in freely behaving small animals," *Journal of Neuroscience Methods,* vol. 155, no. 1, pp. 62-71, 2006.

[55] C.-N. Chien and F.-S. Jaw, "Miniature telemetry system for the recording of action and field potentials," *Journal of Neuroscience Methods,* vol. 147, no. 1, pp. 68-73, 2005.

[56] S. Roy and X. Wang, "Wireless multi-channel single unit recording in freely moving and vocalizing primates," *Journal of Neuroscience Methods,* vol. 203, no. 1, pp. 28-40, 2012.

[57] X. Ye, P. Wang, J. Liu, S. Zhang, J. Jiang, Q. Wang, W. Chen and X. Zheng, "A portable telemetry system for brain stimulation and neuronal activity recording in freely behaving small animals," *Journal of Neuroscience Methods,* vol. 174, no. 2, pp. 186-193, 2008.

[58] R. E. Hampson, V. Collins and S. A. Deadwyler, "A wireless recording system that utilizes Bluetooth technology to transmit neural activity in freely moving animals," *Journal of Neuroscience Methods,* vol. 182, no. 2, pp. 195-204, 2009.

[59] K. Y. Kwon, B. Sirowatka, A. Weber and W. Li, "Opto-uECoG Array: A Hybrid Neural Interface With Transparent uECoG Electrode Array and Integrated LEDs for Optogenetics," *IEEE Transactions on Biomedical Circuits and Systems,* vol. 7, no. 5, pp. 593-600, 2013.

[60] K. Y. Kwon, H.-M. Lee, M. Ghovanloo, A. Weber and W. Li, "A wireless slanted optrode array with integrated micro leds for optogenetics," in *Micro Electro Mechanical Systems (MEMS)*, San Francisco, 2014.

[61] H.-M. Lee, K.-Y. Kwon, W. Li and M. Ghovanloo, "A Wireless Implantable Switched-Capacitor Based Optogenetic Stimulating System," in *Engineering in Medicine and Biology Society (EMBC)*, Chicago, 2014.

[62] M. Chamanzar, M. Borysov, M. Maharbiz and T. Blanche, "High-Density Optrodes for Multi-Scale Electrophysiology and Optogenetic Stimulation," in *Engineering in Medicine and Biology Society (EMBC)*, Chicago, 2014.

[63] C. T. Wentz, J. G. Bernstein, P. Monahan, A. Guerra, A. Rodriguez and E. S. Boyden, "A wirelessly powered and controlled device for optical neural control of freely-behaving animals," *Journal of Neural Engineering,* vol. 8, no. 4, 2011.

[64] "Triangle BioSystems," [Online]. Available: http://www.trianglebiosystems.com/.

[65] "NeuroNexus," [Online]. Available: http://www.neuronexus.com/products/neural-probes/optogenetics/12-neuroscience-products.

[66] "Plexon," [Online]. Available: http://www.plexon.com/products/plexon-solutions-neuroscience-research.

[67] J. Mavoori, A. Jackson, C. Diorio and E. Fetz, "An autonomous implantable computer for neural recording and stimulation in unrestrained primates," *Journal of Neuroscience Methods,* vol. 148, no. 1, pp. 71-77, 2005.

[68] T. Ativanichayaphong, J. W. He, C. E. Hagains, Y. B. Peng and J.-C. Chiao, "A combined wireless neural stimulating and recording system for study of pain processing," *J. Neurosci. Methods,* vol. 170, no. 1, pp. 25-34, 2008.

[69] H. Miranda, V. Gilja, C. A. Chestek, K. Shenoy and T. H. Meng, "HermesD: A High-Rate Long-Range

Wireless Transmission System for Simultaneous Multichannel Neural Recording Applications," *IEEE Transactions on Biomedical Circuits and Systems,* vol. 4, no. 3, pp. 181-191, 2010.

[70] S. P. Gibson, “Neural Spike Sorting in Hardware: From Theory to Practice,” Ph.D. dissertation, Dept. Elect. Eng., University of California, Los Angeles, 2012.

[71] B. Gosselin, M. Sawan and E. Kerherve, "Linear-Phase Delay Filters for Ultra-Low-Power Signal Processing in Neural Recording Implants," *IEEE Transactions on Biomedical Circuits and Systems,* vol. 4, no. 3, pp. 171-180, 2010.

[72] J. Kaiser, "Some useful properties of Teager's energy operators," in *Acoustics, Speech, and Signal Processing*, Minneapolis, 1993.

[73] I. Obeid and P. Wolf, "Evaluation of spike-detection algorithms fora brain-machine interface application," *IEEE Transactions on Biomedical Engineering,* vol. 51, no. 6, pp. 905-911, 2004.

[74] J. Choi and T. Kim, "Neural action potential detector using multi-resolution TEO," *Electronics Letters,* vol. 38, no. 12, pp. 541-543, 2002.

[75] "ROC curve analysis in MedCalc," [Online]. Available: http://www.medcalc.org/manual/roc-curves.php.

[76] M. Abeles and M. J. Goldstein, "Multispike train analysis," *Proc. IEEE,* vol. 65, no. 5, pp. 762-773, 1977.

[77] H. W. Lilliefors, "On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown," *Journal of the American Statistical Association,* vol. 62, no. 318, p. 399–402, 1967.

[78] J. A. Hartigan and P. M. Hartigan, "The Dip Test of Unimodality," *The Annals of Statistics,* vol. 13, no. 1, p. 70–84, 1985.

[79] J. MacQueen, "Some methods for classification and analysis of multivariate observation," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, 1967.

[80] C. Zhang, X. Zhang, M. Q. Zhang and Y. Li, "Neighbor number, valley seeking and clustering," *Pattern Recognition Letters,* vol. 28, no. 2, p. 173–180, 2007.

[81] M. Blatt, S. Wiseman and E. Domany, "Superparamagnetic clustering of data," *Phys Rev Lett.,* vol. 76, no. 18, pp. 3251-3254, 1996.

[82] U. Rutishauser, E. M. Schuman and A. N. Mamelak, "Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo," *Journal of Neuroscience Methods,* vol. 154, no. 1-2, p. 204–224, 2006.

[83] G. Gagnon-Turcotte, C.-O. Dufresne Camaro, A. Avakh, R. Ameli and B. Gosselin, "A Wireless Multichannel Optogenetic Headstage with on-the-Fly Spike Detection," in *International Symposium on Circuits and Systems*, Lisbon, 2015.

[84] C. Chestek, V. Gilja, P. Nuyujukian, S. Ryu, K. Shenoy and R. Kier, "HermesC: RF wireless low-power neural recording system for freely behaving primates," *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 17, no. 4, pp. 1752-1755, 2008.

[85] S. Mirbozorgi, R. Ameli, M. Sawan and B. Gosselin, "Towards a wireless optical stimulation system for long term in-vivo experiments," in *36th annual Conf. of EMBC*, Chicago, 2014, pp. 2024-2027.

[86] "Body weight information," [Online]. Available: http://jaxmice.jax.org/support/weight/000664.html.

[87] "LB G6SP," [Online]. Available: http://www.osram-os.com/osram_os/en/products/product-catalog/led-light-emitting-diodes/advanced-power-topled/lb-g6sp/index.jsp.

[88] "LY G6SP," [Online]. Available: http://www.osram-os.com/osram_os/en/products/product-catalog/led-

light-emitting-diodes/advanced-power-topled/ly-g6sp/index.jsp.

[89] "OSRAM Opto Semiconductor," [Online]. Available: http://www.osram-os.com/osram_os/en/.

[90] "INA118," [Online]. Available: http://www.ti.com/product/ina118.

[91] C. Kitchin and L. Counts, A Designer's Guide to Instrumentation Amplifiers, Analog Devices, 2006.

[92] "LM4140," [Online]. Available: http://www.ti.com/product/lm4140.

[93] K. Kundert, "Power Supply Noise Reduction," [Online]. Available: http://www.designers-guide.org/design/bypassing.pdf.

[94] "MSP430F5328," [Online]. Available: http://www.ti.com/product/msp430f5328.

[95] "FunkOS," [Online]. Available: http://funkos.sourceforge.net/.

[96] "nRF24L01+," [Online]. Available: https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P.

[97] H. W. Ott, Electromagnetic Compatibility Engineering, Wiley , 2009.

[98] "Design Considerations for Mixed-Signal," [Online]. Available: http://www.cn-william.com/uploadfile/Documents/e2v/Application%20Note/Design%20Considerations%20for%20Mixed-Signal.pdf.

[99] "Molex SlimStack Part Number: 53748-0208," [Online]. Available: http://www.molex.com/molex/products/datasheet.jsp?part=active/0537480208_PCB_HEADERS.xml.

[100] "Polymer Lithium Ion Battery - 110mAh," [Online]. Available: https://www.sparkfun.com/products/731.

[101] "Tektronix AFG3101C," [Online]. Available: http://www.testequipmentdepot.com/tektronix/function-generators/afg3101c.htm.

Appendix A. Headstage Prototype Firmware Code

Main Program File (main.c)

The code listing bellow and in the next appendix (page 105) has been created by Gabriel Gagnon-Turcotte based on a code written by Reza Ameli.

```
#include <msp430.h>
#include <stdio.h>
#include "F5529_periph_lib.h"
#include "nRF.h"
#include "driverlib.h"
#include "types.h"
#include "rtos_include.h"
#include "rtos_declaration.h"

// RX and TX buffers. RX is filled by DMA
INT ADC_buffer[ADC_BUFFER_LENGTH];
static BYTE RX_data_buffer[nRF_packet_len];
static INT *ADC_buffer_ptr;
static BYTE UTILS_Packet[nRF_packet_len];

// flags
static BYTE SendTestData_flag;
static ULONG TotalPacketsToSend;
static BIT_QUANTIFICATION quantification;

BYTE dma0_second_buffer = 0;
BYTE dma1_second_buffer = 0;

//Functions definition
void init_system(void);
void send_packet(SEND_PACKET_TYPES type);

//Main
int main(void)
{
    init_system();

    LED_off(LED_GREEN + LED_BLUE);

    __delay_cycles(250000);

    nRF_set_STANDBY1_modeRX(); // The transceiver start in stand-by mode

    //-----------------------------------------------------------------------------
    // miscellaneous
    //-----------------------------------------------------------------------------
    LED_on(LED_GREEN); // IDLE state

    // enable All Interrupts
    __bis_SR_register(GIE);

    //-----------------------------------------------------------------------------
    //-----RTOS INIT-----
    //-----------------------------------------------------------------------------
    Task_Init();

    // Create the application task
    Task_CreateTask(&(stNullTask.stTask),                        // Pointer to the task
                        "null_task",                            // Task name
                        (void*)(stNullTask.ausStack),            // Task stack pointer
                        128,                                    // Task Size
                        0,                                       // Task Priority
```

```
                        (TASK_FUNC)nullTask);                    // Task function pointer

    Task_CreateTask(&(stDataTransmissionTask.stTask),             // Pointer to the task
                    "trans_task",                                // Task name
                    (void*)(stDataTransmissionTask.ausStack),    // Task stack pointer
                    512,                                        // Task Size
                    2,                                            // Task Priority
                    (TASK_FUNC)dataTransmissionTask);            // Task function pointer

    Task_CreateTask(&(stReceptionTask.stTask),                     // Pointer to the task
                    "idle_task",                                // Task name
                    (void*)(stReceptionTask.ausStack),            // Task stack pointer
                    512,                                        // Task Size
                    3,                                            // Task Priority
                    (TASK_FUNC)receptionTask);                    // Task function pointer

    Task_CreateTask(&(stToogleReceptionStandbyTask.stTask),     // Pointer to the task
                    "toog_task",                                // Task name
                    (void*)(stToogleReceptionStandbyTask.ausStack),    // Task stack pointer
                    512,                                        // Task Size
                    4,                                            // Task Priority
                    (TASK_FUNC)toogleReceptionStandbyTask);        // Task function pointer

    Semaphore_Init(&stReceptionSemaphore);
    Semaphore_Init(&stDataTransmissionSemaphore);
    Semaphore_Init(&nRF_IRQ_Semaphore);
    Semaphore_Init(&stTransmitTXBufferSemaphore);
    Semaphore_Init(&stDMASemaphore);

    stReceptionSemaphore.usSem = 1; // Ok to enter IDLE state at starting
    stDataTransmissionSemaphore.usMax = 10;

    Task_Add((TASK_STRUCT*)&stNullTask);
    Task_Add((TASK_STRUCT*)&stReceptionTask);
    Task_Add((TASK_STRUCT*)&stDataTransmissionTask);
    Task_Add((TASK_STRUCT*)&stToogleReceptionStandbyTask);

    Task_Start((TASK_STRUCT*)&stNullTask);                        // Start the tasks
    Task_Start((TASK_STRUCT*)&stReceptionTask);                    // Start the tasks
    Task_Start((TASK_STRUCT*)&stDataTransmissionTask);            // Start the tasks
    Task_Stop((TASK_STRUCT*)&stToogleReceptionStandbyTask);        // Stop the tasks

    KernelSWI_Config();
    KernelSWI_Start();
    KernelTimer_Config();
    KernelTimer_Start();

    //-------------------------------------------------------------------------
    //-----END RTOS INIT-----
    //---------------------------------------------------------------------------

    /*Utils packet header setup*/
    /*Utils packet header*/
    UTILS_Packet[0] = PACKET_UTILS_HEADER_BYTE;
    UTILS_Packet[1] = PACKET_UTILS_HEADER_BYTE;
    UTILS_Packet[2] = PACKET_UTILS_HEADER_BYTE;
    UTILS_Packet[3] = PACKET_UTILS_HEADER_BYTE;
    /*Utils packet tail*/
    UTILS_Packet[28] = PACKET_UTILS_TAIL_BYTE;
    UTILS_Packet[29] = PACKET_UTILS_TAIL_BYTE;
    UTILS_Packet[30] = PACKET_UTILS_TAIL_BYTE;
    UTILS_Packet[31] = PACKET_UTILS_TAIL_BYTE;

    Task_StartTasks();                                // Start the RTOS scheduler
}

//-------------------------------------------------------------------------
//-----RTOS TASKS DEFINITION-----
```

```
//-----------------------------------------------------------------------------
void receptionTask(GENERAL_TASK_STRUCT *pstThis_)
{
    UINT PWM_Freq_Divider;
    BYTE PWM_DutyCycle;
    BYTE k;
    while(1)
    {
        Semaphore_Pend(&stReceptionSemaphore, TIME_FOREVER);                //Wait to enter reception
the state
        Task_Start((TASK_STRUCT*)&stToogleReceptionStandbyTask);            //Start the toggeling
transceiver task
        Semaphore_Pend(&nRF_IRQ_Semaphore,  TIME_FOREVER);                     //Wait for IRQ from the
transceiver
        Task_Stop(&(stToogleReceptionStandbyTask.stTask));                     //Stop the toggeling
transceiver task
        nRF_download_RX_payload(RX_data_buffer);                             // Download the payload
from transceiver
        if(RX_data_buffer[0] == PACKET_ADC_REQUEST)
        {
            __bic_SR_register(GIE);                                          // stop all interrupts
            nRF_ENTER_TRANSMIT_MODE;
            TotalPacketsToSend = ((ULONG)RX_data_buffer[4] << 24) +
                                 ((ULONG)RX_data_buffer[3] << 16) +
                                 ((ULONG)RX_data_buffer[2] << 8) +
                                 ((ULONG)RX_data_buffer[1]);
            // the PWM frequency will be 2MHz divided by this 16-bit value
            PWM_Freq_Divider = ((UINT)RX_data_buffer[6] << 8) + ((UINT)RX_data_buffer[5]);
            // 8-bit number indicating the PWM duty cycle
            PWM_DutyCycle = RX_data_buffer[7];
            if((RX_data_buffer[8] >> 5) & 0x01)
                quantification = BITS_12;
            else
                quantification = BITS_8;
            ADC_setup(quantification);
            // this flag indicates if a know pattern (saw-tooth) should be sent back instead of the real
captured data
            if(RX_data_buffer[8] & 0x0F)
            {
                SendTestData_flag = TRUE;
                for(k = 0 ;k < nRF_packet_len; k++) // dummy data buffer pattern
                    RX_data_buffer[k] = k;
            }
            else
            {
                SendTestData_flag = FALSE;
            }
            // Choose the stimulation LED
            if(RX_data_buffer[9] == 1)
            {
                LED1_PWM_setup(PWM_Freq_Divider);
                LED1_PWM_start( (UINT)(PWM_DutyCycle*((FLOAT)PWM_Freq_Divider/100)));
            }
            else if (RX_data_buffer[9] == 2)
            {
                LED2_PWM_setup(PWM_Freq_Divider);
                LED2_PWM_start( (UINT)(PWM_DutyCycle*((FLOAT)PWM_Freq_Divider/100)));
            }
            LED_off(LED_GREEN);
            LED_on(LED_BLUE);                                                                  // DATA
TRANSMISSION state
            __bis_SR_register(GIE);                                                             // Re-enable
all interrupts
            Task_Sleep_5MS(8);                                                                  // Wait for
the base station to go to receiving mode
            Semaphore_Post(&stDataTransmissionSemaphore);                                      // Go to
transmission state
        }
```

```
        else
        {
            Semaphore_Post(&stReceptionSemaphore); // Still in reception state
        }
    }
}

void dataTransmissionTask(GENERAL_TASK_STRUCT *pstThis_)
{
    volatile ULONG NumberofSentPackets;
    while(1)
    {
        Semaphore_Pend(&stDataTransmissionSemaphore, TIME_FOREVER);                // Wait to enter
transmission state
        KernelTimer_Stop();
        reset_DMA();
        ADC_timer_start();                                                            // Start the
sampling
        Semaphore_Pend(&stTransmitTXBufferSemaphore, TIME_FOREVER);                //Wait for the
buffer to be ready.
        Semaphore_Pend(&stTransmitTXBufferSemaphore, TIME_FOREVER);                //Wait for the
buffer to be ready.
        for(NumberofSentPackets = 1;  NumberofSentPackets <= TotalPacketsToSend; NumberofSentPackets++)
        {
            Semaphore_Pend(&stTransmitTXBufferSemaphore, TIME_FOREVER);            //Wait for the
buffer to be ready.
            if(SendTestData_flag)
                send_packet(SEND_TEST_PACKET);                                        // transmit test
data (saw-tooth pattern)
            else
                send_packet(SEND_ADC_PACKET);                                        // Send ACD raw
data packet
            if(NumberofSentPackets == TotalPacketsToSend)                            // Is it the last
packet?
            {
                ADC_timer_stop();
                wait_for_empty_tx_fifo();                                            // Wait until all
packets are sents
                nRF_CE_low;
                __bic_SR_register(GIE);                                             // stop all
interrupts
                nRF_ENTER_RECEIVE_MODE;
                LED1_PWM_stop();
                LED2_PWM_stop();
                LED_on(LED_GREEN);
                LED_off(LED_BLUE);                                                    // IDLE state
                __bis_SR_register(GIE);                                            // reactivate all
interrupts
            }
        }
        KernelTimer_Start();
        Semaphore_Post(&stReceptionSemaphore);                                        // Go to
reception state
    }
}

void toogleReceptionStandbyTask(GENERAL_TASK_STRUCT *pstThis_)
{
    while(1)
    {
        nRF_ENTER_TRANSMIT_MODE;
        send_packet(SEND_READY_TO_RECEIVE_PACKET);                                    // Notify the
base station that we enter receive mode
        wait_for_empty_tx_fifo();                                                    // Wait until the
packet is sent
        nRF_ENTER_RECEIVE_MODE;                                                    // Start the receive
mode for 50ms
        Task_Sleep_5MS(15);
```

```
        nRF_set_STANDBY1_modeRX();                                                  // Put the
receiver in stand-by mode for 450ms
        Task_Sleep_5MS(95);
    }
}


/*
 * Task designed to save energy when nothing usefull has to be done by other tasks
 */
void nullTask(LIGHT_TASK_STRUCT *pstThis_)
{
    while(1)
    {
        __bis_SR_register(LPM0_bits + GIE);             // Enter LPM0, int enabled
    }
}

//-----------------------------------------------------------------------------
//-----END RTOS TASKS DEFINITION-----
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
//-----UTILITARY FUNCTIONS DEFINITION-----
//-----------------------------------------------------------------------------

void send_packet(SEND_PACKET_TYPES type)
{
    static INT packet_send_counter_counter = 0;
    wait_for_place_in_tx_fifo();                                            //Wait for place in FIFO
    packet_send_counter_counter++;
    switch(type)
    {
        case SEND_ADC_PACKET:
            if(quantification == BITS_8)
                nRF_upload_TX_payload_adc_8(ADC_buffer_ptr);
            else
            {
                nRF_upload_TX_payload_adc_12(ADC_buffer_ptr);
            }
            break;
        case SEND_TEST_PACKET:
            nRF_upload_TX_payload(RX_data_buffer);
            break;
        case SEND_READY_TO_RECEIVE_PACKET:
            UTILS_Packet[4] = PACKET_UTILS_RECEPTION_MODE_BYTE;
            nRF_upload_TX_payload(UTILS_Packet);
            break;
        default:
            break;
    }
    if(packet_send_counter_counter >= 5) //Needs to trigger CE after 4ms, which correspond to 5 packets
    {
        packet_send_counter_counter = 0;
        nRF_CE_low;
        __delay_cycles(200);
    }
    nRF_CE_high;
}



void init_system(void)
{
    WDT_A_hold(WDT_A_BASE); //Stop WDT
    clk_set_8MHz();                 // set CPU clock
    PORT_MAP();                        // alternate functions port map
    Indication_LED_setup();         // indication LEDs
```

```
    SPI_setup();                    // SPI
    ADC_dma_setup();                // DMA
    ADC_setup(BITS_8);                // ADC
    ADC_timer_setup();                // ADC timer, freq = 20,000 Hz
    nRF24L01_pinout_setup();        // Setup the pinout for controling the transceiver
    LED1_PWM_setup(10240);            //Stop all PWM outputs (set 100 hz default value)
    LED2_PWM_setup(10240);
    LED1_PWM_start(0);
    LED2_PWM_start(0);
}

//-----------------------------------------------------------------------
//-----END UTILITARY FUNCTIONS DEFINITION-----
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
//-----INTERRUPTS DEFINITIONS-----
//-----------------------------------------------------------------------

/*
 * nRF24L01+ IRQ
 */
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    volatile BYTE status;
    P1IFG &= ~BIT6;                                                 // IFG cleared
    status = nRF_NOP();                                             // Looks for what caused the IRQ
    nRF_clear_IRQ();                                                 // Reset the IRQ
    if(status & 0x20) //TX interrupt?
    {
        __delay_cycles(1); //Debug point
    }
    else                                                             //We received a packet
    {
        if((status & 0x40) != 0)
        {
            Semaphore_Post(&nRF_IRQ_Semaphore); //Unlock the IDLE state
        }
        else
        {
            __delay_cycles(1); // Debug point
        }
    }
}

/*
 * Interrupt called when the ADC buffer is full
 */
#pragma vector=DMA_VECTOR
__interrupt void DMA_ISR (void)
{
    switch(__even_in_range(DMAIV,16))
    {
        case  0: break;                         // No interrupt
        case  1: break;                         // No interrupt
        case  2:
            DMA_enableTransfers(DMA_BASE, DMA_CHANNEL_1); // Enable DMA1
            switch(dma0_second_buffer)
            {
            case 0:
                dma0_second_buffer = 1;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (ULONG)&ADC_buffer[32],
DMA_DIRECTION_INCREMENT);
                break;
            case 1:
                dma0_second_buffer = 2;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (ULONG)&ADC_buffer[64],
```

```
DMA_DIRECTION_INCREMENT);
                break;
            case 2:
                dma0_second_buffer = 3;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (ULONG)&ADC_buffer[96],
DMA_DIRECTION_INCREMENT);
                break;
            case 3:
                dma0_second_buffer = 4;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (ULONG)&ADC_buffer[128],
DMA_DIRECTION_INCREMENT);
                break;
            case 4:
                dma0_second_buffer = 5;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (ULONG)&ADC_buffer[160],
DMA_DIRECTION_INCREMENT);
                break;
            case 5:
                dma0_second_buffer = 0;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (ULONG)&ADC_buffer[0],
DMA_DIRECTION_INCREMENT);
                break;
            default:
                break;
            }
            break;
        case  4:
            DMA_enableTransfers(DMA_BASE, DMA_CHANNEL_0); // Enable DMA0
            switch(dma1_second_buffer)
            {
            case 0:
                dma1_second_buffer = 1;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (ULONG)&ADC_buffer[48],
DMA_DIRECTION_INCREMENT);
                ADC_buffer_ptr = &ADC_buffer[0];
                Semaphore_Post(&stTransmitTXBufferSemaphore);
                break;
            case 1:
                dma1_second_buffer = 2;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (ULONG)&ADC_buffer[80],
DMA_DIRECTION_INCREMENT);
                ADC_buffer_ptr = &ADC_buffer[32];
                Semaphore_Post(&stTransmitTXBufferSemaphore);
                break;
            case 2:
                dma1_second_buffer = 3;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (ULONG)&ADC_buffer[112],
DMA_DIRECTION_INCREMENT);
                ADC_buffer_ptr = &ADC_buffer[64];
                Semaphore_Post(&stTransmitTXBufferSemaphore);
                break;
            case 3:
                dma1_second_buffer = 4;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (ULONG)&ADC_buffer[144],
DMA_DIRECTION_INCREMENT);
                ADC_buffer_ptr = &ADC_buffer[96];
                Semaphore_Post(&stTransmitTXBufferSemaphore);
                break;
            case 4:
                dma1_second_buffer = 5;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (ULONG)&ADC_buffer[176],
DMA_DIRECTION_INCREMENT);
                ADC_buffer_ptr = &ADC_buffer[128];
                Semaphore_Post(&stTransmitTXBufferSemaphore);
                break;
            case 5:
                dma1_second_buffer = 0;
                DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (ULONG)&ADC_buffer[16],
```

```
DMA_DIRECTION_INCREMENT);
                ADC_buffer_ptr = &ADC_buffer[160];
                Semaphore_Post(&stTransmitTXBufferSemaphore);
                break;
            default:
                break;
            }
            break;                              // DMA1IFG
        case  6: break;                         // DMA2IFG
        case  8: break;                         // Reserved
        case 10: break;                         // Reserved
        case 12: break;                         // Reserved
        case 14: break;                         // Reserved
        case 16: break;                         // Reserved
        default: break;
    }
}
/*If we get here, there is a huge problem going on*/
#pragma vector=UNMI_VECTOR
__interrupt void NMI_ISR(void)
{
    volatile uint16_t status;
    do {
            // If it still can't clear the oscillator fault flags after the timeout,
            // trap and wait here.
            status = UCS_clearAllOscFlagsWithTimeout(UCS_BASE, 1000);
    } while (status != 0);
}

//-----------------------------------------------------------------------------
//-----END INTERRUPTS DEFINITIONS-----
//-----------------------------------------------------------------------------
```

Custom Peripheral Control Library Header File (F5529_periph_lib.h)

```
#ifndef F5132_PERIPH_LIB_H_
#define F5132_PERIPH_LIB_H_

#include <msp430.h>
#include "utils.h"

#define LED_GREEN BIT0
#define LED_BLUE BIT1

void clk_set_8MHz(void);
void UART_setup(void);
void SPI_setup(void);
void nRF24L01_pinout_setup(void);

// Indication LEDs
void Indication_LED_setup(void);
void LED_on(char LED);     // LED can be either LED_BLUE or LED_GREEN or both
void LED_off(char LED);    // LED can be either LED_BLUE or LED_GREEN or both
void LED_toggle(char LED); // LED can be either LED_BLUE or LED_GREEN or both

// LED PWM
void PORT_MAP(void);
void LED1_PWM_setup(unsigned int clock_prescaler);
void LED2_PWM_setup(unsigned int clock_prescaler);
void LED1_PWM_start(unsigned int duty_cycle);
void LED1_PWM_stop(void);
void LED2_PWM_start(unsigned int duty_cycle);
void LED2_PWM_stop(void);

// Analog-to-digital converter and its timer
void ADC_setup(BIT_QUANTIFICATION quantification);
void ADC_timer_setup();
void ADC_timer_start(void);
void ADC_timer_stop(void);
unsigned char read_first_ADC_Channel(void);
unsigned char read_second_ADC_Channel(void);
void ADC_dma_setup(void);
void ADC_dma_stop(void);
void reset_DMA(void);

#endif /* F5132_PERIPH_LIB_H_ */
```

Custom Peripheral Control Library Implementation File (F5529_periph_lib.c)

```
#include "F5529_periph_lib.h"
#include "kernelcfg.h"
#include "driverlib.h"
#include "nRF.h"


extern int ADC_buffer[ADC_BUFFER_LENGTH];
extern char dma0_second_buffer;
extern char dma1_second_buffer;

//---------------------------------------------------------------------------
// System clock setup
//---------------------------------------------------------------------------
void clk_set_8MHz(void)
{
    volatile uint32_t clockValue = 0;
    //Set DCO FLL reference = REFO
    UCS_clockSignalInit(UCS_BASE, UCS_FLLREF, UCS_REFOCLK_SELECT, UCS_CLOCK_DIVIDER_1);
    //Set ACLK = REFO
    UCS_clockSignalInit(UCS_BASE, UCS_ACLK, UCS_REFOCLK_SELECT, UCS_CLOCK_DIVIDER_1);
    //Set core configuration
    PMM_setVCore(PMM_BASE, PMM_CORE_LEVEL_3);
    PMM_SvsLDisabledInLPMFastWake(PMM_BASE);
    PMM_SvsHEnabledInLPMFullPerf(PMM_BASE);
    PMM_SvsLEnabledInLPMFastWake(PMM_BASE);
    //Set Ratio and Desired MCLK Frequency  and initialize DCO
    UCS_initFLLSettle(UCS_BASE, 8000, 250);
    // Enable global oscillator fault flag
    SFR_clearInterrupt(SFR_BASE, SFR_OSCILLATOR_FAULT_INTERRUPT);
    SFR_enableInterrupt(SFR_BASE,SFR_OSCILLATOR_FAULT_INTERRUPT);
    // Enable global interrupt
    __bis_SR_register(GIE);
    clockValue = UCS_getSMCLK(UCS_BASE);         // Debug point
    clockValue = UCS_getMCLK(UCS_BASE);         // Debug point
    clockValue = UCS_getACLK(UCS_BASE);         // Debug point
}


//---------------------------------------------------------------------------
// UART for system test
//---------------------------------------------------------------------------
void UART_setup(void)
{
    //P3.3, P3.4 = USCI_A0 TXD/RXD
    GPIO_setAsPeripheralModuleFunctionInputPin(
            GPIO_PORT_P3,
            GPIO_PIN3 + GPIO_PIN4
            );

    USCI_A_UART_initAdvance(USCI_A0_BASE,
            USCI_A_UART_CLOCKSOURCE_SMCLK,
            8,
            0,
            3,
            USCI_A_UART_NO_PARITY,
            USCI_A_UART_LSB_FIRST,
            USCI_A_UART_ONE_STOP_BIT,
            USCI_A_UART_MODE,
            USCI_A_UART_AUTOMATIC_BAUDRATE_DETECTION_MODE
            );

    USCI_A_UART_enable(USCI_A0_BASE);

    //Enable Receive Interrupt
    USCI_A_UART_clearInterruptFlag(USCI_A0_BASE,
```

```
                                    USCI_A_UART_RECEIVE_INTERRUPT
                                    );
    USCI_A_UART_enableInterrupt(USCI_A0_BASE,
                                USCI_A_UART_RECEIVE_INTERRUPT
                                );
}
//-----------------------------------------------------------------------------
// nRF24L01+ pins configuration. CE, IRQ pins and SS.
//-----------------------------------------------------------------------------
void nRF24L01_pinout_setup(void)
{
    //Set P1.5 to output direction, nRF_SS
    GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN5);
    GPIO_setDriveStrength(GPIO_PORT_P1, GPIO_PIN5, GPIO_REDUCED_OUTPUT_DRIVE_STRENGTH);
    //Set P4.0 to output direction, nRF_CE
    GPIO_setAsOutputPin(GPIO_PORT_P4, GPIO_PIN0);
    //Set P1.6 to input direction, nRF_IRQ
    GPIO_setAsInputPin(GPIO_PORT_P1, GPIO_PIN6);
    // Enable nRF_IRQ interrupt
    GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN6);
    // hi-to-low interrupt
    GPIO_interruptEdgeSelect(GPIO_PORT_P1, GPIO_PIN6, GPIO_HIGH_TO_LOW_TRANSITION);
    // clear nRF_IRQ interrupt
    GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN6);
}
//-----------------------------------------------------------------------------
// SPI for Nordic transceiver and the PGA
//-----------------------------------------------------------------------------
void SPI_setup(void)
{
    // SPI: UCB0
    // Clock freq = SMCLK
    // Idle clock polarity = low
    // Data is captured on the rising (first) clock edge
    // CLK = P1.3, MISO = P1.5, MOSI = P1.4, Nordic Slave-Select = P2.5, AFE Slave-Select = P2.6

    //P4.1, P4.2, P4.3 option select
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4, GPIO_PIN2);
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P4,  GPIO_PIN1 + GPIO_PIN3);
    //Initialize Master
    USCI_B_SPI_masterInit(USCI_B1_BASE,
                          USCI_B_SPI_CLOCKSOURCE_SMCLK,
                          UCS_getSMCLK(UCS_BASE),
                          8192000,
                          USCI_B_SPI_MSB_FIRST,
                          USCI_B_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT,
                          USCI_B_SPI_CLOCKPOLARITY_INACTIVITY_LOW
                          );

    //Enable SPI module
    USCI_B_SPI_enable(USCI_B1_BASE);
}

//-----------------------------------------------------------------------------
// Indication LEDs
//-----------------------------------------------------------------------------
void Indication_LED_setup(void)
{
    //Set P1.0 to output direction
    GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0);

    //Set P4.7 to output direction
    GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1);
}

void LED_on(char LED)
{
    if(LED & BIT0)
```

```
        GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);
    if(LED & BIT1)
        GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);
}
void LED_off(char LED)
{
    if(LED & BIT0)
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
    if(LED & BIT1)
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
}
void LED_toggle(char LED)
{
    if(LED & BIT0)
        GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN0);
    if(LED & BIT1)
        GPIO_toggleOutputOnPin(GPIO_PORT_P2, GPIO_PIN1);
}
//------------------------------------------------------------------------
void PORT_MAP(void)
{
    //Port mapping for the PWM
    const uint8_t port_mapping[] = {
                //Port P4:
                PM_NONE,
                PM_UCB1SIMO,
                PM_UCB1SOMI,
                PM_UCB1CLK,
                PM_NONE,
                PM_NONE,
                PM_TB0CCR1A,
                PM_TB0CCR2A,
        };
    // Reconfigure P4, PWM on P4.1 and P4.2
    PMAP_configurePorts(PMAP_CTRL_BASE,
                        (const uint8_t*)port_mapping,
                        (uint8_t*)&P4MAP01,
                        1,
                        PMAP_DISABLE_RECONFIGURATION
                        );
}
//------------------------------------------------------------------------
// Optical stimulation LEDs
//------------------------------------------------------------------------
void LED1_PWM_setup(unsigned int clock_prescaler)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P4, GPIO_PIN7);

    //Start Up Timer
    TIMER_B_configureUpMode( TIMER_B0_BASE,
                             TIMER_B_CLOCKSOURCE_SMCLK,
                             TIMER_B_CLOCKSOURCE_DIVIDER_8,
                             clock_prescaler-1,
                             TIMER_B_TBIE_INTERRUPT_DISABLE,
                             TIMER_B_CCIE_CCR0_INTERRUPT_DISABLE,
                             TIMER_B_SKIP_CLEAR
                             );
}

void LED2_PWM_setup(unsigned int clock_prescaler)
{
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P4, GPIO_PIN6);

    //Start Up Down Timer
    TIMER_B_configureUpMode( TIMER_B0_BASE,
                             TIMER_B_CLOCKSOURCE_SMCLK,
                             TIMER_B_CLOCKSOURCE_DIVIDER_8,
                             clock_prescaler-1,
```

```
                            TIMER_B_TBIE_INTERRUPT_DISABLE,
                            TIMER_B_CCIE_CCR0_INTERRUPT_DISABLE,
                            TIMER_B_DO_CLEAR
                            );
}

void LED1_PWM_start(unsigned int duty_cycle)
{
    //Generate PWM 1
    TIMER_B_initCompare(TIMER_B0_BASE,
                        TIMER_B_CAPTURECOMPARE_REGISTER_1,
                        TIMER_B_CAPTURECOMPARE_INTERRUPT_DISABLE,
                        TIMER_B_OUTPUTMODE_TOGGLE_SET | TIMER_B_OUTPUTMODE_OUTBITVALUE,
                        duty_cycle
                        );

    TIMER_B_startCounter(TIMER_B0_BASE,
                            TIMER_B_UP_MODE
                            );
}
void LED1_PWM_stop(void)
{
    //Stop PWM 1
    TIMER_B_initCompare(TIMER_B0_BASE,
                            TIMER_B_CAPTURECOMPARE_REGISTER_1,
                            TIMER_B_CAPTURECOMPARE_INTERRUPT_DISABLE,
                            TIMER_B_OUTPUTMODE_OUTBITVALUE,
                            0
                            );
}
void LED2_PWM_start(unsigned int duty_cycle)
{
    //Generate PWM 2
    TIMER_B_initCompare(TIMER_B0_BASE,
                        TIMER_B_CAPTURECOMPARE_REGISTER_2,
                        TIMER_B_CAPTURECOMPARE_INTERRUPT_DISABLE,
                        TIMER_B_OUTPUTMODE_TOGGLE_SET | TIMER_B_OUTPUTMODE_OUTBITVALUE,
                        duty_cycle
                        );

    TIMER_B_startCounter(TIMER_B0_BASE,
                         TIMER_B_UP_MODE
                         );
}
void LED2_PWM_stop(void)
{
    //Stop PWM 2
    TIMER_B_initCompare(TIMER_B0_BASE,
                            TIMER_B_CAPTURECOMPARE_REGISTER_2,
                            TIMER_B_CAPTURECOMPARE_INTERRUPT_DISABLE,
                            TIMER_B_OUTPUTMODE_OUTBITVALUE,
                            0
                            );
}
//==============================================================================
void ADC_setup(BIT_QUANTIFICATION quantification)
{
    static char ref_set = FALSE;
    if(!ref_set)
    {
        ref_set = TRUE;
        while(REF_isRefGenBusy(REF_BASE));
        REF_setReferenceVoltage(REF_BASE, REF_VREF2_5V);
        REF_enableReferenceVoltage(REF_BASE);
    }

    // Enable A/D channel inputs on P6.0, P6.1, P6.2 and P6.3
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6, GPIO_PIN0 + GPIO_PIN1 + GPIO_PIN2 +
```

```
GPIO_PIN3);

    // ADC12 clock source SMCLK, triggered by timer A0 CCR1
    ADC12_A_init(ADC12_A_BASE,
                ADC12_A_SAMPLEHOLDSOURCE_1,
                ADC12_A_CLOCKSOURCE_SMCLK,
                ADC12_A_CLOCKDIVIDER_2
                );

    ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                               ADC12_A_CYCLEHOLD_32_CYCLES,
                               ADC12_A_CYCLEHOLD_32_CYCLES,
                               ADC12_A_MULTIPLESAMPLESDISABLE);

    ADC12_A_disableReferenceBurst(ADC12_A_BASE);
    ADC12_A_setReferenceBufferSamplingRate(ADC12_A_BASE,
                                           ADC12_A_MAXSAMPLINGRATE_200KSPS);

    if(quantification == BITS_8)
        ADC12_A_setResolution(ADC12_A_BASE, ADC12_A_RESOLUTION_8BIT);
    else
        ADC12_A_setResolution(ADC12_A_BASE, ADC12_A_RESOLUTION_12BIT);

    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_INPUT_A0,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_1,
                            ADC12_A_INPUT_A1,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_2,
                            ADC12_A_INPUT_A2,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_3,
                            ADC12_A_INPUT_A3,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_4,
                            ADC12_A_INPUT_A0,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_5,
                            ADC12_A_INPUT_A1,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_6,
                            ADC12_A_INPUT_A2,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_7,
                            ADC12_A_INPUT_A3,
```

```
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_8,
                            ADC12_A_INPUT_A0,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_9,
                            ADC12_A_INPUT_A1,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_10,
                            ADC12_A_INPUT_A2,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_11,
                            ADC12_A_INPUT_A3,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_12,
                            ADC12_A_INPUT_A0,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_13,
                            ADC12_A_INPUT_A1,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_14,
                            ADC12_A_INPUT_A2,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_NOTENDOFSEQUENCE);
    ADC12_A_memoryConfigure(ADC12_A_BASE,
                            ADC12_A_MEMORY_15,
                            ADC12_A_INPUT_A3,
                            ADC12_A_VREFPOS_INT,
                            ADC12_A_VREFNEG_AVSS,
                            ADC12_A_ENDOFSEQUENCE);

    ADC12_A_enable(ADC12_A_BASE);
}


void ADC_timer_setup(BIT_QUANTIFICATION quantification)
{

    //Configure timer A0. Source SMCLK, CCR0 = SYSTEM_FREQ / SPECIAL_FUNCTION_FREQ
    TIMER_A_configureUpMode( TIMER_A0_BASE,
                             TIMER_A_CLOCKSOURCE_SMCLK,
                             TIMER_A_CLOCKSOURCE_DIVIDER_1,
                             SYSTEM_FREQ / SPECIAL_FUNCTION_FREQ,
                             TIMER_A_TAIE_INTERRUPT_DISABLE,
                             TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE,
                             TIMER_A_DO_CLEAR
                             );
```

```
    //Configure timer A0 CCR1 to trigger the ADC12 at CCR1 = SYSTEM_FREQ / SPECIAL_FUNCTION_FREQ
    TIMER_A_initCompare(TIMER_A0_BASE,
                        TIMER_A_CAPTURECOMPARE_REGISTER_1,
                        TIMER_A_CAPTURECOMPARE_INTERRUPT_DISABLE,
                        TIMER_A_OUTPUTMODE_SET_RESET,
                        SYSTEM_FREQ / SPECIAL_FUNCTION_FREQ
                        );
}

void ADC_timer_start(void)
{
    DMA_enableTransfers(DMA_BASE, DMA_CHANNEL_0);
    ADC12_A_enable(ADC12_A_BASE); // Start ADC at the same time
    ADC12_A_startConversion(ADC12_A_BASE,
                            ADC12_A_MEMORY_0,
                            ADC12_A_REPEATED_SEQOFCHANNELS);
    TIMER_A_startCounter(TIMER_A0_BASE, TIMER_A_UP_MODE);
}

void ADC_timer_stop(void)
{
    DMA_disableTransfers(DMA_BASE, DMA_CHANNEL_0);
    DMA_disableTransfers(DMA_BASE, DMA_CHANNEL_1);
    DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (unsigned long)&ADC_buffer[0], DMA_DIRECTION_INCREMENT);
    DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (unsigned long)&ADC_buffer[16], DMA_DIRECTION_INCREMENT);
    TIMER_A_stop(TIMER_A0_BASE);
}

void ADC_dma_setup(void)
{
    //-----Channel 0 config-----
    DMA_init(DMA_BASE,
             DMA_CHANNEL_0,
             DMA_TRANSFER_BURSTBLOCK,
             16,
             DMA_TRIGGERSOURCE_24,
             DMA_SIZE_SRCWORD_DSTWORD,
             DMA_TRIGGER_RISINGEDGE);

    DMA_setSrcAddress(DMA_BASE, DMA_CHANNEL_0, (unsigned long)&ADC12MEM0, DMA_DIRECTION_INCREMENT);

    DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (unsigned long)&ADC_buffer[16], DMA_DIRECTION_INCREMENT);

    DMA_enableTransfers(DMA_BASE, DMA_CHANNEL_0);

    DMA_enableInterrupt(DMA_BASE, DMA_CHANNEL_0);

    //-----Channel 1 config-----
    DMA_init(DMA_BASE,
             DMA_CHANNEL_1,
             DMA_TRANSFER_BURSTBLOCK,
             16,
             DMA_TRIGGERSOURCE_24,
             DMA_SIZE_SRCWORD_DSTWORD,
             DMA_TRIGGER_RISINGEDGE);

    DMA_setSrcAddress(DMA_BASE, DMA_CHANNEL_1, (unsigned long)&ADC12MEM0, DMA_DIRECTION_INCREMENT);

    DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (unsigned long)&ADC_buffer[16], DMA_DIRECTION_INCREMENT);

    DMA_enableInterrupt(DMA_BASE, DMA_CHANNEL_1);
}

void ADC_dma_stop(void)
{
    DMA_disableTransfers(DMA_BASE, DMA_CHANNEL_0);
    DMA_disableTransfers(DMA_BASE, DMA_CHANNEL_1);
    DMA_disableTransfers(DMA_BASE, DMA_CHANNEL_2);
```

```
}

void reset_DMA(void)
{
    dma0_second_buffer = 0;
    dma1_second_buffer = 0;
    DMA_setSrcAddress(DMA_BASE, DMA_CHANNEL_0, (unsigned long)&ADC12MEM0, DMA_DIRECTION_INCREMENT);
    DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_0, (unsigned long)&ADC_buffer[0], DMA_DIRECTION_INCREMENT);
    DMA_enableTransfers(DMA_BASE, DMA_CHANNEL_0);

    DMA_setSrcAddress(DMA_BASE, DMA_CHANNEL_1, (unsigned long)&ADC12MEM0, DMA_DIRECTION_INCREMENT);
    DMA_setDstAddress(DMA_BASE, DMA_CHANNEL_1, (unsigned long)&ADC_buffer[16], DMA_DIRECTION_INCREMENT);
    DMA_disableTransfers(DMA_BASE, DMA_CHANNEL_1);
}
```

Custom nRF24L01+ Controller Library Header File (nRF.h)

```
#include <msp430.h>
#include "F5529_periph_lib.h"
#include "utils.h"

extern void nRF_init(void);
extern void nRF_reg_write(char addr, char *data, unsigned data_length);
extern void nRF_reg_read(char addr, char *data, unsigned data_length);
extern void nRF_upload_TX_payload_adc_8(int *data);
extern void nRF_upload_TX_payload_adc_12(int *data);
extern void nRF_upload_TX_payload(char *data);
extern void nRF_download_RX_payload(char *data);
extern void nRF_FLUSH_TX(void);
extern void nRF_FLUSH_RX(void);
extern char nRF_NOP(void);
extern char nRF_FIFO_STATUS(void);
extern void nRF_clear_IRQ(void);
extern char nRF_read_RX_payload_len(void);
extern void nRF_set_TX_mode(void);
extern void nRF_set_RX_mode(void);
extern void nRF_set_STANDBY1_modeRX(void);
extern void wait_for_place_in_tx_fifo(void);
extern void wait_for_empty_tx_fifo(void);

//-------------------------------------------------------------------
#define nRF_DESELECT while(UCB1STAT & UCBUSY); P1OUT |= BIT5
#define nRF_SELECT P1OUT &= ~BIT5

#define nRF_CE_high P4OUT |= BIT0
#define nRF_CE_low P4OUT &= ~BIT0

#define PULSE_CE nRF_CE_high; __delay_cycles(200); nRF_CE_low // The delay value must be set according
to the CPU clock frequency and must be at least 10 microseconds


#define nRF_ENTER_RECEIVE_MODE nRF_CE_low; \
                            nRF_init(); \
                            nRF_set_RX_mode(); \
                            nRF_clear_IRQ(); \
                            nRF_CE_high

#define nRF_ENTER_TRANSMIT_MODE nRF_CE_low; \
                                nRF_init(); \
                                nRF_clear_IRQ(); \
                                nRF_set_TX_mode()

//-------------------------------------------------------------------
```

Custom nRF24L01+ Controller Library Implementation File (nRF.c)

```
#include "driverlib.h"
#include "nRF.h"
//-----------------------------------------------------------------------
char tmp;
void nRF_init(void)
{
    unsigned int i;
    nRF_CE_low;
    for(i = 0 ; i < 65000; i++);
    nRF_SELECT;
    for(i = 0 ; i < 65000; i++);
    nRF_DESELECT;
}
//-----------------------------------------------------------------------
void nRF_reg_write(char addr, char *data, unsigned data_length)
{
    char i;

    addr = addr & 0x1F;
    addr = addr | 0x20;

    nRF_SELECT;

    UCB1TXBUF = addr;

    for(i = 0; i < data_length; i++)
    {
        while(UCB1STAT & UCBUSY);
        UCB1TXBUF = data[i];
    }

    while(UCB1STAT & UCBUSY);
    nRF_DESELECT;

    //SPI_buffer_flush();
}
//-----------------------------------------------------------------------
void nRF_reg_read(char addr, char *data, unsigned data_length)
{
    char i;

    addr = addr & 0x1F;

    //SPI_buffer_flush();

    nRF_SELECT;

    UCB1TXBUF = addr;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    for(i = 0; i < data_length; i++)
    {
        UCB1TXBUF = 0xFF;
        while(UCB1STAT & UCBUSY);
        *(data+i) = UCB1RXBUF;
    }

    while(UCB1STAT & UCBUSY);
    nRF_DESELECT;
}
//-----------------------------------------------------------------------
void nRF_upload_TX_payload(char *data)
{
    char i;
```

```
    __bic_SR_register(GIE);  // stop all interrupts

    nRF_SELECT;

    UCB1TXBUF =  0xA0;

    for(i = 0; i < nRF_packet_len; i++)
    {
        while(UCB1STAT & UCBUSY);
        UCB1TXBUF = data[i];
    }

    while(UCB1STAT & UCBUSY);
    nRF_DESELECT;

    __bis_SR_register(GIE); // reactivate all interrupts

    //SPI_buffer_flush();
}
//-----------------------------------------------------------------------
void nRF_upload_TX_payload_adc_8(int *data)
{
    char i;

    __bic_SR_register(GIE);  // stop all interrupts

    nRF_SELECT;

    UCB1TXBUF =  0xA0;

    for(i = 0; i < nRF_packet_len; i++)
    {
        while(UCB1STAT & UCBUSY);
        UCB1TXBUF = data[i];
    }

    while(UCB1STAT & UCBUSY);
    nRF_DESELECT;

    __bis_SR_register(GIE); // reactivate all interrupts
}
//-----------------------------------------------------------------------
void nRF_upload_TX_payload_adc_12(int *data)
{
    char i;
    char *ptr = (char*)data;

    __bic_SR_register(GIE);  // stop all interrupts

    nRF_SELECT;

    UCB1TXBUF =  0xA0;

    for(i = 0; i < 2*nRF_packet_len; i += 8)
    {
        while(UCB1STAT & UCBUSY);
        UCB1TXBUF = ptr[i];
        while(UCB1STAT & UCBUSY);
        UCB1TXBUF = ptr[i + 1] + (ptr[i + 4] & 0xF0);
        while(UCB1STAT & UCBUSY);
        UCB1TXBUF = ptr[i + 2];
        while(UCB1STAT & UCBUSY);
        UCB1TXBUF = ptr[i + 3] + (ptr[i + 6] & 0xF0);
    }

    while(UCB1STAT & UCBUSY);
    nRF_DESELECT;
```

```
    __bis_SR_register(GIE); // reactivate all interrupts
}
//-----------------------------------------------------------------------
void nRF_download_RX_payload(char *data)
{
    char i;

    //SPI_buffer_flush();

    nRF_SELECT;

    UCB1TXBUF =  0x61;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    for(i = 0; i < nRF_packet_len; i++)
    {
        UCB1TXBUF = 0xFF;
        while(UCB1STAT & UCBUSY);
        *(data+i) = UCB1RXBUF;
    }

    while(UCB1STAT & UCBUSY);
    nRF_DESELECT;
}
//-----------------------------------------------------------------------
void nRF_FLUSH_TX(void)
{
    nRF_SELECT;

    UCB1TXBUF =  0xE1;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    nRF_DESELECT;
}
//-----------------------------------------------------------------------
void nRF_FLUSH_RX(void)
{
    nRF_SELECT;

    UCB1TXBUF =  0xE2;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    nRF_DESELECT;
}
//-----------------------------------------------------------------------
char nRF_NOP(void)
{
    char status;

    nRF_SELECT;

    UCB1TXBUF = 0xFF;
    while(UCB1STAT & UCBUSY);
    status = UCB1RXBUF;

    nRF_DESELECT;

    return status;
}
//-----------------------------------------------------------------------
char nRF_FIFO_STATUS(void)
{
    char status;
```

```
    nRF_SELECT;

    UCB1TXBUF =  0x17;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    UCB1TXBUF = 0xFF;
    while(UCB1STAT & UCBUSY);
    status = UCB1RXBUF;

    nRF_DESELECT;

    return status;
}
//-------------------------------------------------------------------
char nRF_read_RX_payload_len(void)
{
    char length;

    nRF_SELECT;

    UCB1TXBUF =  0x60;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    UCB1TXBUF = 0xFF;
    while(UCB1STAT & UCBUSY);
    length = UCB1RXBUF;

    nRF_DESELECT;

    return length;
}
//-------------------------------------------------------------------
void nRF_clear_IRQ(void)
{
    nRF_SELECT;

    UCB1TXBUF =  0x27;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    UCB1TXBUF =  0x70;
    while(UCB1STAT & UCBUSY);
    tmp = UCB1RXBUF;

    nRF_DESELECT;
}
//-------------------------------------------------------------------
void nRF_set_STANDBY1_modeRX(void)
{
    char data =0x19;
    nRF_CE_high;
    nRF_reg_write(0x00, &data, 1);
}
//-------------------------------------------------------------------
void wait_for_place_in_tx_fifo(void)
{
    volatile char fifo_status;
    fifo_status = nRF_FIFO_STATUS();
    while(fifo_status & 0x20)
    {
        __delay_cycles(100);
        fifo_status = nRF_FIFO_STATUS();
    }
}
//-------------------------------------------------------------------
void wait_for_empty_tx_fifo(void)
```

```
{
    volatile char fifo_status;
    fifo_status = nRF_FIFO_STATUS(); // Check the fifo status of the transceiver
    while(!(fifo_status & 0x10))
    {
        __delay_cycles(100);
        fifo_status = nRF_FIFO_STATUS();
    }
}
```

Custom nRF24L01+ Controller Library Implementation File (nRF_config.c)

```
#include "nRF.h"

//-----------------------------------
// nRF24L01+ TX mode settings
void nRF_set_TX_mode(void)
{
    char data[5];
    unsigned int i;
    //++++++++++++++++++++++++++
    // CONFIG : Configuration Register
    data[0] = 0x1A;
    nRF_reg_write(0x00,data,1);
    //++++++++++++++++++++++++++
    for(i = 0 ; i < 65000; i++);
    //++++++++++++++++++++++++++
    // EN_AA : Enhanced ShockBurst
    data[0] = 0x00;
    nRF_reg_write(0x01,data,1);
    //++++++++++++++++++++++++++
    // EN_RXADDR : Enabled RX Addresses
    data[0] = 0x01;
    nRF_reg_write(0x02,data,1);
    //++++++++++++++++++++++++++
    // SETUP_AW : Setup of Address Widths
    data[0] = 0x01;
    nRF_reg_write(0x03,data,1);
    //++++++++++++++++++++++++++
    // SETUP_RETR : Setup of Automatic Retransmission
    data[0] = 0x00;
    nRF_reg_write(0x04,data,1);
    //++++++++++++++++++++++++++
    // RF_CH : RF Channel
    data[0] = 0x65;
    nRF_reg_write(0x05,data,1);
    //++++++++++++++++++++++++++
    // RF_SETUP : RF Setup Register
    data[0] = 0x0F;
    nRF_reg_write(0x06,data,1);
    //++++++++++++++++++++++++++
    // STATUS : Status Register
    //++++++++++++++++++++++++++
    // OBSERVE_TX : Transmit observe register
    //++++++++++++++++++++++++++
    // RPD : Received Power Detector
    //++++++++++++++++++++++++++
    // RX_ADDR_P0 : Receive address data pipe 0
    data[0] = 0x0F;
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x0A,data,5);
    //++++++++++++++++++++++++++
    // RX_ADDR_P1 : Receive address data pipe 1
    //++++++++++++++++++++++++++
    // RX_ADDR_P2 : Receive address data pipe 2
    //++++++++++++++++++++++++++
    // RX_ADDR_P3 : Receive address data pipe 3
    //++++++++++++++++++++++++++
    // RX_ADDR_P4 : Receive address data pipe 4
    //++++++++++++++++++++++++++
    // RX_ADDR_P5 : Receive address data pipe 5
    //++++++++++++++++++++++++++
    // TX_ADDR : Transmit address
    data[0] = 0x0F;
```

```
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x10,data,5);
    //++++++++++++++++++++++++++
    // RX_PW_P0 : Number of bytes in RX payload in data pipe 0 ???????????
    data[0] = nRF_packet_len;
    nRF_reg_write(0x11,data,1);
    //++++++++++++++++++++++++++
    // RX_PW_P1 : Number of bytes in RX payload in data pipe 1
    //++++++++++++++++++++++++++
    // RX_PW_P2 : Number of bytes in RX payload in data pipe 2
    //++++++++++++++++++++++++++
    // RX_PW_P3 : Number of bytes in RX payload in data pipe 3
    //++++++++++++++++++++++++++
    // RX_PW_P4 : Number of bytes in RX payload in data pipe 4
    //++++++++++++++++++++++++++
    // RX_PW_P5 : Number of bytes in RX payload in data pipe 5
    //++++++++++++++++++++++++++
    // FIFO_STATUS : FIFO Status Register
    //++++++++++++++++++++++++++
    // DYNPD : Enable dynamic payload length
    data[0] = 0x00;
    nRF_reg_write(0x1C,data,1);
    //++++++++++++++++++++++++++
    // FEATURE : Feature Register
    data[0] = 0x00;
    nRF_reg_write(0x1D,data,1);
    //++++++++++++++++++++++++++
}
//-----------------------------------
// nRF24L01+ RX mode settings
void nRF_set_RX_mode(void)
{
    char data[5];
    unsigned int i;
    //++++++++++++++++++++++++++
    // CONFIG : Configuration Register
    data[0] = 0x1B;
    nRF_reg_write(0x00,data,1);
    //++++++++++++++++++++++++++
    for(i = 0 ; i < 65000; i++);
    //++++++++++++++++++++++++++
    // EN_AA : Enhanced ShockBurst
    data[0] = 0x00;
    nRF_reg_write(0x01,data,1);
    //++++++++++++++++++++++++++
    // EN_RXADDR : Enabled RX Addresses
    data[0] = 0x01;
    nRF_reg_write(0x02,data,1);
    //++++++++++++++++++++++++++
    // SETUP_AW : Setup of Address Widths
    data[0] = 0x01;
    nRF_reg_write(0x03,data,1);
    //++++++++++++++++++++++++++
    // SETUP_RETR : Setup of Automatic Retransmission
    data[0] = 0x00;
    nRF_reg_write(0x04,data,1);
    //++++++++++++++++++++++++++
    // RF_CH : RF Channel
    data[0] = 0x0A;
    nRF_reg_write(0x05,data,1);
    //++++++++++++++++++++++++++
    // RF_SETUP : RF Setup Register
    data[0] = 0x0E;
    nRF_reg_write(0x06,data,1);
    //++++++++++++++++++++++++++
```

```
    // STATUS : Status Register
    //++++++++++++++++++++++++++
    // OBSERVE_TX : Transmit observe register
    //++++++++++++++++++++++++++
    // RPD : Received Power Detector
    //++++++++++++++++++++++++++
    // RX_ADDR_P0 : Receive address data pipe 0
    data[0] = 0x0F;
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x0A,data,5);
    //++++++++++++++++++++++++++
    // RX_ADDR_P1 : Receive address data pipe 1
    //++++++++++++++++++++++++++
    // RX_ADDR_P2 : Receive address data pipe 2
    //++++++++++++++++++++++++++
    // RX_ADDR_P3 : Receive address data pipe 3
    //++++++++++++++++++++++++++
    // RX_ADDR_P4 : Receive address data pipe 4
    //++++++++++++++++++++++++++
    // RX_ADDR_P5 : Receive address data pipe 5
    //++++++++++++++++++++++++++
    // TX_ADDR : Transmit address
    data[0] = 0x0F;
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x10,data,5);
    //++++++++++++++++++++++++++
    // RX_PW_P0 : Number of bytes in RX payload in data pipe 0 ???????????
    data[0] = nRF_packet_len;
    nRF_reg_write(0x11,data,1);
    //++++++++++++++++++++++++++
    // RX_PW_P1 : Number of bytes in RX payload in data pipe 1
    //++++++++++++++++++++++++++
    // RX_PW_P2 : Number of bytes in RX payload in data pipe 2
    //++++++++++++++++++++++++++
    // RX_PW_P3 : Number of bytes in RX payload in data pipe 3
    //++++++++++++++++++++++++++
    // RX_PW_P4 : Number of bytes in RX payload in data pipe 4
    //++++++++++++++++++++++++++
    // RX_PW_P5 : Number of bytes in RX payload in data pipe 5
    //++++++++++++++++++++++++++
    // FIFO_STATUS : FIFO Status Register
    //++++++++++++++++++++++++++
    // DYNPD : Enable dynamic payload length
    data[0] = 0x00;
    nRF_reg_write(0x1C,data,1);
    //++++++++++++++++++++++++++
    // FEATURE : Feature Register
    data[0] = 0x00;
    nRF_reg_write(0x1D,data,1);
    //++++++++++++++++++++++++++
}
//-------------------------------------
```

Appendix B. Baseband Prototype Firmware Code

Main Program Header File (MAIN.h)

```
#define PART_TM4C1233H6PM
#define rvmdk

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/ssi.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include "driverlib/systick.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib/timer.h"
//---------------------------------------------
extern void ConfigureGPIO(void);
extern void ConfigureSysClock(void);
extern void ConfigureUART(void);
extern void ConfigureSysTick(uint32_t period);
extern void ConfigureSPI(void);
extern void ConfigureSPI_DAC(void);
extern void Write_DAC(int pData);
extern void ConfigureDAC_Timer(void);
extern void SPI_buffer_flush(void);
//---------------------------------------------
extern void LED_blink_red(void);
extern void LED_blink_blue(void);
extern void LED_blink_green(void);
extern void delay(uint32_t val);
//---------------------------------------------
extern void SysTick_int_handler(void);
extern void UART1_int_handler(void);
extern void nRF_IRQ_handler(void);
extern void Timer0IntHandler(void);
//---------------------------------------------
#define DAC_CH1 0x1000
#define DAC_CH2 0x9000
#define DAC_C_B_Size 256

#define LED_on_red GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1)
#define LED_off_red GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0)

#define LED_on_blue GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2)
#define LED_off_blue GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0)

#define LED_on_green GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3)
#define LED_off_green GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0)

#define SWITCH_RECEIVE_MODE IntMasterDisable();\
                                                        nRF_init();\
                                                        nRF_write_PRX_settings();\
                                                        nRF_clear_IRQ();\
                                                        nRF_CE_high;\
```

```
                                                        GPIOIntClear(nRF_IRQn_PORT, nRF_IRQn_PIN);\
                                                        IntMasterEnable()

#define SWITCH_TRANSMIT_MODE IntMasterDisable();\
                                                         nRF_init();\
                                                         nRF_write_PTX_settings();\
                                                         nRF_clear_IRQ();\
                                                         nRF_CE_low;\
                                                         GPIOIntClear(nRF_IRQn_PORT, nRF_IRQn_PIN);\
                                                         IntMasterEnable();

#define SET_IDLE_STATE  LED_on_blue;\
                                                LED_off_red;\
                                                UART_Data_Received = false;\
                                                Current_State = STATE_IDLE;

#define SET_RECEPTION_STATE LED_off_blue;\
                                                        LED_on_red;\
                                                        Current_State = STATE_Data_Reception;

#define UTILS_PACKET Data_Buffer[0] == 0xCC && Data_Buffer[1] == 0xCC && Data_Buffer[2] == 0xCC \
                                            && Data_Buffer[3] == 0xCC && Data_Buffer[28] == 0xDD &&
Data_Buffer[29] == 0xDD \
                                                && Data_Buffer[30] == 0xDD && Data_Buffer[31] == 0xDD
//---------------------------------------------
typedef enum
{
    STATE_IDLE,
    STATE_Data_Reception,
    STATE_Switch_to_IDLE
}STATES;

typedef enum
{
    BITS_8,
    BITS_12
}QUANTIFICATION;
//---------------------------------------------
```

Main Program File (main.c)

```
#include <MAIN.h>
#include <nRF.h>

static char UARTCommand_Index = 0;
static bool UART_Data_Received = false;
static bool UARTCommand_Available = false;
static bool nRF_IRQ = false;
static bool spike_mode;
static uint32_t UARTCommand[32];
static uint32_t Data_Buffer[32];
static uint32_t DAC_Circular_Buffer[DAC_C_B_Size];
static uint32_t DAC_Circular_Buffer_Spike_1[DAC_C_B_Size];
static uint32_t DAC_Circular_Buffer_Spike_2[DAC_C_B_Size];
static uint32_t DAC_C_B_Index_w;
static uint32_t DAC_C_B_Index_r;
static uint32_t DAC_C_B_Index_Spike_1_w;
static uint32_t DAC_C_B_Index_Spike_1_r;
static uint32_t DAC_C_B_Index_Spike_2_w;
static uint32_t DAC_C_B_Index_Spike_2_r;
static uint32_t total_number_of_packets;
static uint32_t number_of_received_packets;
static STATES Current_State = STATE_IDLE;
static QUANTIFICATION Bits_Quantification = BITS_8;

int main(void)
{
        uint32_t i;
        char fifo_status;
        unsigned char uart_data_received;

    //-----System configuration-----
    ConfigureSysClock();
    ConfigureGPIO();
    ConfigureUART();
    ConfigureSPI();
        ConfigureSPI_DAC();
        ConfigureDAC_Timer();
        //-----Ens system configuration-----

        //-----DAC configuration-----
        DAC_C_B_Index_w = 0;
        DAC_C_B_Index_r = 0;
        DAC_C_B_Index_Spike_1_r = 0;
        DAC_C_B_Index_Spike_1_w = 0;
        DAC_C_B_Index_Spike_2_r = 0;
        DAC_C_B_Index_Spike_2_r = 0;
        Write_DAC(0x1C00);
        Write_DAC(0x9C00);
        //-----End DAC configuration-----

    nRF_init();
    nRF_clear_IRQ();
    nRF_write_PTX_settings();
    nRF_CE_low;
    //-----------------------------------------------
    for(i = 0; i < 32; i++)
    {
        Data_Buffer[i] = 0;
        UARTCommand[i] = 0;
    }

        SET_IDLE_STATE;
    //-----------------------------------------------
    IntMasterEnable();
    //-----------------------------------------------
```

```
    while(1)
    {
        switch(Current_State)
        {
            //-----------------------------------------
            case STATE_IDLE:
                if(UART_Data_Received == true)
                {
                    UART_Data_Received = false;
                                        uart_data_received = UARTCharGet(UART1_BASE);
                                        if(!(UARTCommand_Index == 0 && uart_data_received == 0xDD))
                                        {
                                            UARTCommand[UARTCommand_Index++] = uart_data_received;
                                            if(UARTCommand_Index == 10)
                                            {
                                                    UARTCommand_Index = 0;
                                                    UARTCommand_Available = true;
                                            }
                                        }
                }
                if(UARTCommand_Available == true)
                {
                    UARTCommand_Available = false;

                    total_number_of_packets = ((unsigned long)UARTCommand[4] << 24) +
                                              ((unsigned long)UARTCommand[3] << 16) +
                                              ((unsigned long)UARTCommand[2] << 8) +
                                              (unsigned long)UARTCommand[1];
                    number_of_received_packets = 0;

                                        if((UARTCommand[8] >> 4) & 0x01)
                                            spike_mode = true;
                                        else
                                            spike_mode = false;

                                        if((UARTCommand[8] >> 5) & 0x01)
                                            Bits_Quantification = BITS_12;
                                        else
                                            Bits_Quantification = BITS_8;

                                        /*
                                        The headstage will notify us when we can send the configuration
packet
                                        We have to wait for his notification packet
                                        */
                                        SWITCH_RECEIVE_MODE;

                                        while(1)
                                        {
                                                while(!nRF_IRQ);
                                                nRF_clear_IRQ();
                                                nRF_download_RX_payload(Data_Buffer);
                                                nRF_IRQ = false;
                                                nRF_clear_IRQ();
                                                if(UTILS_PACKET)
                                                {
                                                        if(Data_Buffer[4] == 0x00)
                                                        {
                                                                break; // Notification packet received,
OK to send the configuration packet
                                                        }
                                                }
                                        }

                                        SWITCH_TRANSMIT_MODE;

                    SysCtlDelay(3000000); // Wait for the receiver to go in receive mode
```

```
                    nRF_upload_TX_payload(UARTCommand);  // Send the configuration packet
                    nRF_CE_high;
                    SysCtlDelay(150);
                    nRF_CE_low;

                    while(!nRF_IRQ);
                    nRF_IRQ = false;
                    nRF_clear_IRQ();

                    SWITCH_RECEIVE_MODE;

                    SET_RECEPTION_STATE;
                }
            break;
            //-----------------------------------------
            case STATE_Data_Reception:
                                fifo_status = nRF_FIFO_STATUS();
                if(!(fifo_status & 0x01))
                {
                    nRF_clear_IRQ();
                    nRF_download_RX_payload(Data_Buffer);
                    nRF_IRQ = false;

                                        // Verify if the packet received is a utils packet or a
notitication packet
                                        if(UTILS_PACKET)
                                        {
                                                if(Data_Buffer[4] == 0x01)
                                                {
                                                        UARTCharPut(UART1_BASE, 0xBB); // Send the utils
byte to the base station, notify that the data following data are not from ADC
                                                }
                                                else if(Data_Buffer[4] == 0x02)
                                                {
                                                        SWITCH_TRANSMIT_MODE
                                                        SET_IDLE_STATE;
                                                        continue;
                                                }
                                                else if(Data_Buffer[4] == 0x00 &&
number_of_received_packets == 0)
                                                {
                                                      // We received a ok to send notification packet.
This means that the configuration packet that we send get lost.
                                                        // We need to resend the configuration packet
                                                        UARTCommand_Available = true;
                                                        SET_IDLE_STATE;
                                                        continue;
                                                }
                                                else
                                                {
                                                        continue;  // Low probability that we get here,
if so, nothing happens
                                                }
                                        }
                                        else
                                        {
                                                number_of_received_packets++;
                                                if(spike_mode)
                                                {
                                                    UARTCharPut(UART1_BASE, 0xCC); // Send the data byte
to the base station, notify that the data following are from the ADC of the headstage
                                                    if(Data_Buffer[0] == 0x01)
                                                    {
                                                        for(i = 1; i < 32; i++)
                                                        {
```

```
DAC_Circular_Buffer_Spike_1[DAC_C_B_Index_Spike_1_w++] = Data_Buffer[i];
                                                                    if(DAC_C_B_Index_Spike_1_w >=
DAC_C_B_Size)
                                                                        DAC_C_B_Index_Spike_1_w = 0;
                                                                }
                                                            }
                                                            else if (Data_Buffer[0] == 0x02)
                                                            {
                                                                for(i = 1; i < 32; i++)
                                                                {


DAC_Circular_Buffer_Spike_2[DAC_C_B_Index_Spike_2_w++] = Data_Buffer[i];
                                                                    if(DAC_C_B_Index_Spike_2_w >=
DAC_C_B_Size)
                                                                        DAC_C_B_Index_Spike_2_w = 0;
                                                                }

                                                            }
                                                        }
                                                        else
                                                        {
                                                            if(Bits_Quantification == BITS_8)
                                                                UARTCharPut(UART1_BASE, 0xAA);  // Send the data
byte to the base station, notify that the data following are from the ADC of the headstage
                                                            else
                                                                UARTCharPut(UART1_BASE, 0xEE);
                                                            for(i = 0; i < 32; i += 4)
                                                            {
                                                                    if(Bits_Quantification == BITS_8)
                                                                    {// 8 bits
                                                                        DAC_Circular_Buffer[DAC_C_B_Index_w++] =
Data_Buffer[i];

                                                                        if(DAC_C_B_Index_w >= DAC_C_B_Size)
                                                                            DAC_C_B_Index_w = 0;


                                                                        DAC_Circular_Buffer[DAC_C_B_Index_w++] =
Data_Buffer[i + 1];

                                                                        if(DAC_C_B_Index_w >= DAC_C_B_Size)
                                                                            DAC_C_B_Index_w = 0;
                                                                    }
                                                                    else
                                                                    {// 12 bits

DAC_Circular_Buffer[DAC_C_B_Index_w++] = Data_Buffer[i] + ((uint32_t)(Data_Buffer[i + 1] & 0x0F) << 8);
                                                                        if(DAC_C_B_Index_w >= DAC_C_B_Size)
                                                                            DAC_C_B_Index_w = 0;

DAC_Circular_Buffer[DAC_C_B_Index_w++] = Data_Buffer[i + 2] + ((uint32_t)(Data_Buffer[i + 3] & 0x0F) <<
8);
                                                                        if(DAC_C_B_Index_w >= DAC_C_B_Size)
                                                                            DAC_C_B_Index_w = 0;
                                                                    }
                                                            }
                                                        }
                                                    }

                        for(i = 0; i<32; i++)
                            UARTCharPut(UART1_BASE, Data_Buffer[i]);

                        if(number_of_received_packets == total_number_of_packets && spike_mode == false)
                        {
                            Current_State = STATE_Switch_to_IDLE;
                        }
                    }
                break;
                //-----------------------------------------
                            case STATE_Switch_to_IDLE:
```

```
                        {
                                SWITCH_TRANSMIT_MODE;
                                SET_IDLE_STATE;
                        }
                        break;
        }
    }
}

void SysTick_int_handler(void)
{
}

void nRF_IRQ_handler(void)
{
    GPIOIntClear(nRF_IRQn_PORT, nRF_IRQn_PIN);
    nRF_IRQ = true;
}

void UART1_int_handler(void)
{
        volatile unsigned char header_byte;
    UARTIntClear(UART1_BASE, UART_INT_RX | UART_INT_RT);
    if(Current_State == STATE_Data_Reception)
    {
                // We received new configuration data to send while we were already receiving data from
the headstage
                header_byte = UARTCharGet(UART1_BASE);
        if(header_byte == 0xDE)  // New data, switch to IDLE state
        {
                        UARTCommand_Index = 1;
                        UARTCommand[0] = 0xDE;  // Simulate that we were already in IDLE state
            Current_State = STATE_Switch_to_IDLE;
        }
                else if(header_byte == 0xDD) // only switch to IDLE state
        {
                        UARTCommand_Index = 0;
            Current_State = STATE_Switch_to_IDLE;
                }
    }
    else if(Current_State == STATE_IDLE)
        UART_Data_Received = true;

}

void Timer0IntHandler(void)
{
    // Clear the timer interrupt.
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
        if(Current_State == STATE_Data_Reception)
        {
            if(spike_mode)
            {
                if(DAC_Circular_Buffer_Spike_1[DAC_C_B_Index_Spike_1_r] != 0x8000)
                {
                    Write_DAC(DAC_CH1 | (DAC_Circular_Buffer_Spike_1[DAC_C_B_Index_Spike_1_r] << 2));

                    DAC_Circular_Buffer_Spike_1[DAC_C_B_Index_Spike_1_r] = 0x8000;

                    DAC_C_B_Index_Spike_1_r += 1;
                    if(DAC_C_B_Index_Spike_1_r >= DAC_C_B_Size)
                        DAC_C_B_Index_Spike_1_r = 0;
                }
                if(DAC_Circular_Buffer_Spike_2[DAC_C_B_Index_Spike_2_r] != 0x8000)
                {
                    Write_DAC(DAC_CH2 | (DAC_Circular_Buffer_Spike_2[DAC_C_B_Index_Spike_2_r] << 2));

                    DAC_Circular_Buffer_Spike_2[DAC_C_B_Index_Spike_2_r] = 0x8000;
```

```
                    DAC_C_B_Index_Spike_2_r += 1;
                    if(DAC_C_B_Index_Spike_2_r >= DAC_C_B_Size)
                        DAC_C_B_Index_Spike_2_r = 0;
                }
            }
            else
            {
                if(DAC_Circular_Buffer[DAC_C_B_Index_r + 1] != 0x8000)
                {
                    if(Bits_Quantification == BITS_8)
                    {
                        Write_DAC(DAC_CH1 | (DAC_Circular_Buffer[DAC_C_B_Index_r] << 4));
                        Write_DAC(DAC_CH2 | (DAC_Circular_Buffer[DAC_C_B_Index_r + 1] << 4));
                    }
                    else
                    {
                        Write_DAC(DAC_CH1 | (DAC_Circular_Buffer[DAC_C_B_Index_r]));
                        Write_DAC(DAC_CH2 | (DAC_Circular_Buffer[DAC_C_B_Index_r + 1]));
                    }

                    DAC_Circular_Buffer[DAC_C_B_Index_r] = 0x8000;
                    DAC_Circular_Buffer[DAC_C_B_Index_r + 1] = 0x8000;

                    DAC_C_B_Index_r += 2;
                    if(DAC_C_B_Index_r >= DAC_C_B_Size)
                        DAC_C_B_Index_r = 0;
                }
            }
        }
        else
        {
            Write_DAC(DAC_CH1 | 0);
            Write_DAC(DAC_CH2 | 0);
        }
}
```

Custom nRF24L01+ Controller Library Header File (nRF.h)

```
extern void nRF_init(void);

extern void nRF_reg_write(char addr, uint32_t *data, unsigned data_length);
extern void nRF_reg_read(char addr, uint32_t *data, unsigned data_length);

extern void nRF_upload_TX_payload(uint32_t *data);
extern void nRF_download_RX_payload(uint32_t *data);

extern void nRF_FLUSH_TX(void);
extern void nRF_FLUSH_RX(void);

extern char nRF_FIFO_STATUS(void);
extern uint32_t nRF_NOP(void);

extern void nRF_clear_IRQ(void);

extern uint32_t nRF_read_RX_payload_len(void);

extern void nRF_write_PTX_settings(void);
extern void nRF_write_PRX_settings(void);
//-------------------------------------------------------------------
#define nRF_CE_PORT GPIO_PORTE_BASE
#define nRF_CE_PIN GPIO_PIN_1

#define nRF_CSN_PORT GPIO_PORTE_BASE
#define nRF_CSN_PIN GPIO_PIN_4

#define nRF_IRQn_PORT GPIO_PORTA_BASE
#define nRF_IRQn_PIN GPIO_PIN_7

#define nRF_packet_len 32
//-------------------------------------------------------------------
#define nRF_DESELECT while(SSIBusy(SSI0_BASE));GPIOPinWrite(nRF_CSN_PORT, nRF_CSN_PIN, nRF_CSN_PIN)
#define nRF_SELECT GPIOPinWrite(GPIO_PORTE_BASE, nRF_CSN_PIN, 0)

#define nRF_CE_high GPIOPinWrite(nRF_CE_PORT, nRF_CE_PIN, nRF_CE_PIN)
#define nRF_CE_low GPIOPinWrite(nRF_CE_PORT, nRF_CE_PIN, 0)
//-------------------------------------------------------------------
```

Custom nRF24L01+ Controller Library Implementation File (nRF.c)

```
#include <MAIN.h>
#include <nRF.h>
//-------------------------------------------------------------------
void nRF_init(void)
{
    nRF_CE_low;
    delay(100000);
    nRF_SELECT;
    delay(100000);
    nRF_DESELECT;
}
//-------------------------------------------------------------------
void nRF_reg_write(char addr, uint32_t *data, unsigned data_length)
{
    uint32_t i;

    addr = addr & 0x1F;
    addr = addr | 0x20;

    nRF_SELECT;

    SSIDataPut(SSI0_BASE, addr);

    for(i = 0; i < data_length; i++)
    {
        while(SSIBusy(SSI0_BASE));
        SSIDataPut(SSI0_BASE, data[i]);
    }

    while(SSIBusy(SSI0_BASE));
    nRF_DESELECT;

    SPI_buffer_flush();
}
//-------------------------------------------------------------------
void nRF_reg_read(char addr, uint32_t *data, unsigned data_length)
{
    uint32_t i;

    addr = addr & 0x1F;

    SPI_buffer_flush();

    nRF_SELECT;

    SSIDataPut(SSI0_BASE, addr);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, NULL);

    for(i = 0; i < data_length; i++)
    {
        SSIDataPut(SSI0_BASE, 0xFF);
        while(SSIBusy(SSI0_BASE));
        SSIDataGet(SSI0_BASE, data+i);
    }

    while(SSIBusy(SSI0_BASE));
    nRF_DESELECT;
}
//-------------------------------------------------------------------
void nRF_upload_TX_payload(uint32_t *data)
{
    uint32_t i;

    nRF_SELECT;
```

```
    SSIDataPut(SSI0_BASE, 0xA0);

    for(i = 0; i < nRF_packet_len; i++)
    {
        while(SSIBusy(SSI0_BASE));
        SSIDataPut(SSI0_BASE, data[i]);
    }

    while(SSIBusy(SSI0_BASE));
    nRF_DESELECT;

    SPI_buffer_flush();
}
//-----------------------------------------------------------------------
void nRF_download_RX_payload(uint32_t *data)
{
    uint32_t i;

    SPI_buffer_flush();

    nRF_SELECT;

    SSIDataPut(SSI0_BASE, 0x61);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, NULL);

    for(i = 0; i < nRF_packet_len; i++)
    {
        SSIDataPut(SSI0_BASE, 0xFF);
        while(SSIBusy(SSI0_BASE));
        SSIDataGet(SSI0_BASE, data+i);
    }

    while(SSIBusy(SSI0_BASE));
    nRF_DESELECT;
}
//-----------------------------------------------------------------------
void nRF_FLUSH_TX(void)
{
    nRF_SELECT;

    SSIDataPut(SSI0_BASE, 0xE1);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, NULL);

    nRF_DESELECT;
}
//-----------------------------------------------------------------------
void nRF_FLUSH_RX(void)
{
    nRF_SELECT;

    SSIDataPut(SSI0_BASE, 0xE2);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, NULL);

    nRF_DESELECT;
}
//-----------------------------------------------------------------------
uint32_t nRF_NOP(void)
{
    uint32_t status;

    nRF_SELECT;

    SSIDataPut(SSI0_BASE, 0xFF);
    while(SSIBusy(SSI0_BASE));
```

```
    SSIDataGet(SSI0_BASE, &status);

    nRF_DESELECT;

    return status;
}
//-------------------------------------------------------------------
uint32_t nRF_read_RX_payload_len(void)
{
    uint32_t length;

    nRF_SELECT;

    SSIDataPut(SSI0_BASE, 0x60);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, NULL);

    SSIDataPut(SSI0_BASE, 0xFF);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, &length);

    nRF_DESELECT;

    return length;
}
//-------------------------------------------------------------------
void nRF_clear_IRQ(void)
{
    nRF_SELECT;

    SSIDataPut(SSI0_BASE, 0x27);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, NULL);

    SSIDataPut(SSI0_BASE, 0x70);
    while(SSIBusy(SSI0_BASE));
    SSIDataGet(SSI0_BASE, NULL);

    nRF_DESELECT;
}
//-------------------------------------------------------------------
char nRF_FIFO_STATUS(void)
{
    volatile uint32_t status;

    nRF_SELECT;

    SSIDataPut(SSI0_BASE, 0x17);
        while(SSIBusy(SSI0_BASE)){}
    SSIDataGet(SSI0_BASE, NULL);

        SSIDataPut(SSI0_BASE, 0xFF);
    while(SSIBusy(SSI0_BASE)){}
    SSIDataGet(SSI0_BASE, (uint32_t*)&status);

    nRF_DESELECT;

    return (char)status;
}
```

Custom nRF24L01+ Controller Library Implementation File (nRF_config.c)

```c
#include <MAIN.h>
#include <nRF.h>

//-----------------------------------
// nRF24L01+ PTX register settings
void nRF_write_PTX_settings(void)
{
    uint32_t data[5];

    unsigned int i;
    //++++++++++++++++++++++++++++
    // CONFIG : Configuration Register
    data[0] = 0x1A;
    nRF_reg_write(0x00,data,1);
    //++++++++++++++++++++++++++++
    for(i = 0 ; i < 65000; i++);
    //++++++++++++++++++++++++++++
    // EN_AA : Enhanced ShockBurst
    data[0] = 0x00;
    nRF_reg_write(0x01,data,1);
    //++++++++++++++++++++++++++++
    // EN_RXADDR : Enabled RX Addresses
    data[0] = 0x01;
    nRF_reg_write(0x02,data,1);
    //++++++++++++++++++++++++++++
    // SETUP_AW : Setup of Address Widths
    data[0] = 0x01;
    nRF_reg_write(0x03,data,1);
    //++++++++++++++++++++++++++++
    // SETUP_RETR : Setup of Automatic Retransmission
    data[0] = 0x00;
    nRF_reg_write(0x04,data,1);
    //++++++++++++++++++++++++++++
    // RF_CH : RF Channel
    data[0] = 0x0A;
    nRF_reg_write(0x05,data,1);
    //++++++++++++++++++++++++++++
    // RF_SETUP : RF Setup Register
    data[0] = 0x0E;
    nRF_reg_write(0x06,data,1);
    //++++++++++++++++++++++++++++
    // STATUS : Status Register
    //++++++++++++++++++++++++++++
    // OBSERVE_TX : Transmit observe register
    //++++++++++++++++++++++++++++
    // RPD : Received Power Detector
    //++++++++++++++++++++++++++++
    // RX_ADDR_P0 : Receive address data pipe 0
    data[0] = 0x0F;
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x0A,data,5);
    //++++++++++++++++++++++++++++
    // RX_ADDR_P1 : Receive address data pipe 1
    //++++++++++++++++++++++++++++
    // RX_ADDR_P2 : Receive address data pipe 2
    //++++++++++++++++++++++++++++
    // RX_ADDR_P3 : Receive address data pipe 3
    //++++++++++++++++++++++++++++
    // RX_ADDR_P4 : Receive address data pipe 4
    //++++++++++++++++++++++++++++
    // RX_ADDR_P5 : Receive address data pipe 5
    //++++++++++++++++++++++++++++
```

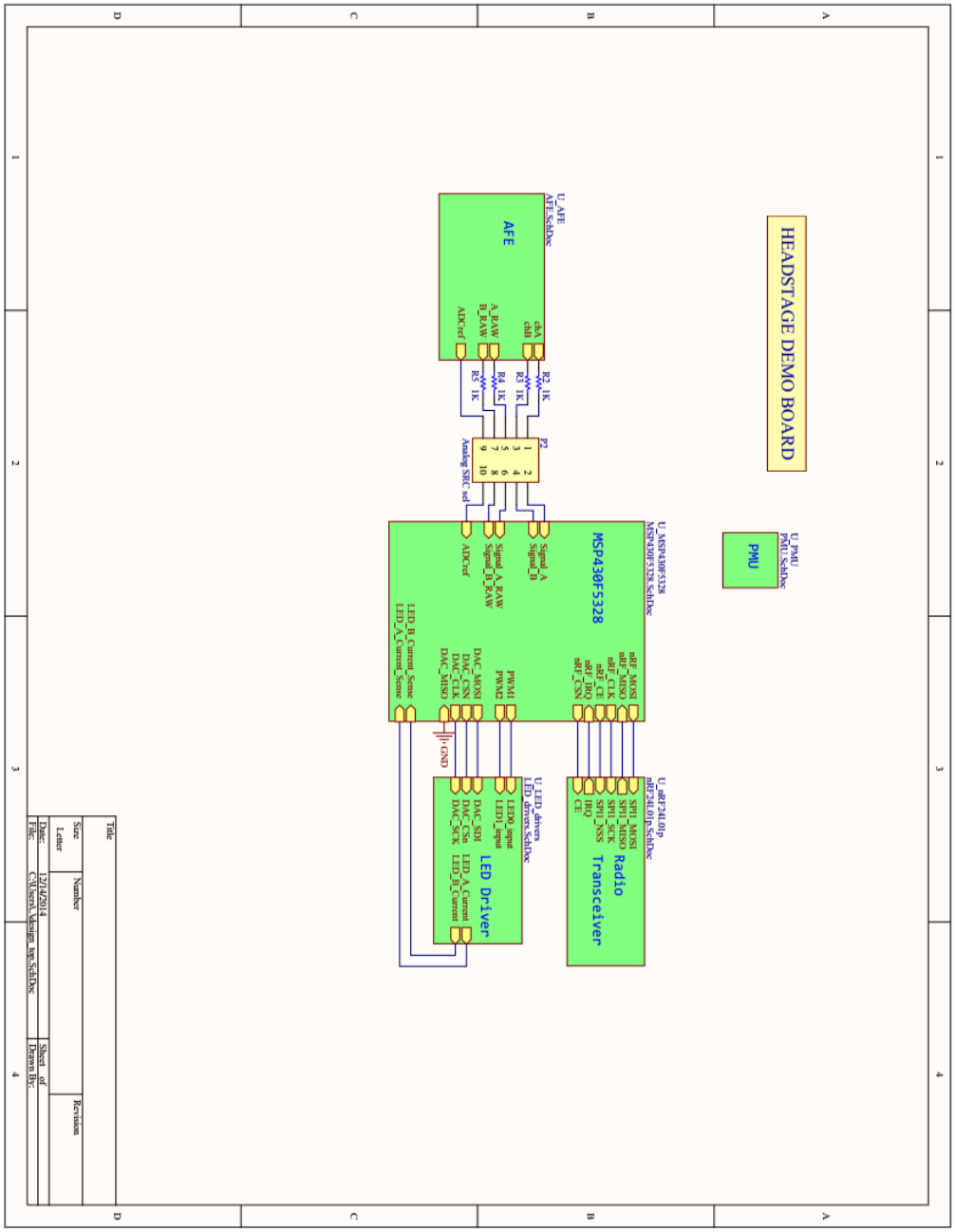
```
    // TX_ADDR : Transmit address
    data[0] = 0x0F;
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x10,data,5);
    //++++++++++++++++++++++++++
    // RX_PW_P0 : Number of bytes in RX payload in data pipe 0 ???????????
    data[0] = nRF_packet_len;
    nRF_reg_write(0x11,data,1);
    //++++++++++++++++++++++++++
    // RX_PW_P1 : Number of bytes in RX payload in data pipe 1
    //++++++++++++++++++++++++++
    // RX_PW_P2 : Number of bytes in RX payload in data pipe 2
    //++++++++++++++++++++++++++
    // RX_PW_P3 : Number of bytes in RX payload in data pipe 3
    //++++++++++++++++++++++++++
    // RX_PW_P4 : Number of bytes in RX payload in data pipe 4
    //++++++++++++++++++++++++++
    // RX_PW_P5 : Number of bytes in RX payload in data pipe 5
    //++++++++++++++++++++++++++
    // FIFO_STATUS : FIFO Status Register
    //++++++++++++++++++++++++++
    // DYNPD : Enable dynamic payload length
    data[0] = 0x00;
    nRF_reg_write(0x1C,data,1);
    //++++++++++++++++++++++++++
    // FEATURE : Feature Register
    data[0] = 0x00;
    nRF_reg_write(0x1D,data,1);
    //++++++++++++++++++++++++++
}
//-----------------------------------
// nRF24L01+ PRX register settings
void nRF_write_PRX_settings(void)
{
   uint32_t data[5];

    unsigned int i;
    //++++++++++++++++++++++++++
    // CONFIG : Configuration Register
    data[0] = 0x1B;
    nRF_reg_write(0x00,data,1);
    //++++++++++++++++++++++++++
    for(i = 0 ; i < 65000; i++);
    //++++++++++++++++++++++++++
    // EN_AA : Enhanced ShockBurst
    data[0] = 0x00;
    nRF_reg_write(0x01,data,1);
    //++++++++++++++++++++++++++
    // EN_RXADDR : Enabled RX Addresses
    data[0] = 0x01;
    nRF_reg_write(0x02,data,1);
    //++++++++++++++++++++++++++
    // SETUP_AW : Setup of Address Widths
    data[0] = 0x01;
    nRF_reg_write(0x03,data,1);
    //++++++++++++++++++++++++++
    // SETUP_RETR : Setup of Automatic Retransmission
    data[0] = 0x00;
    nRF_reg_write(0x04,data,1);
    //++++++++++++++++++++++++++
    // RF_CH : RF Channel
    data[0] = 0x65;
    nRF_reg_write(0x05,data,1);
    //++++++++++++++++++++++++++
    // RF_SETUP : RF Setup Register
```

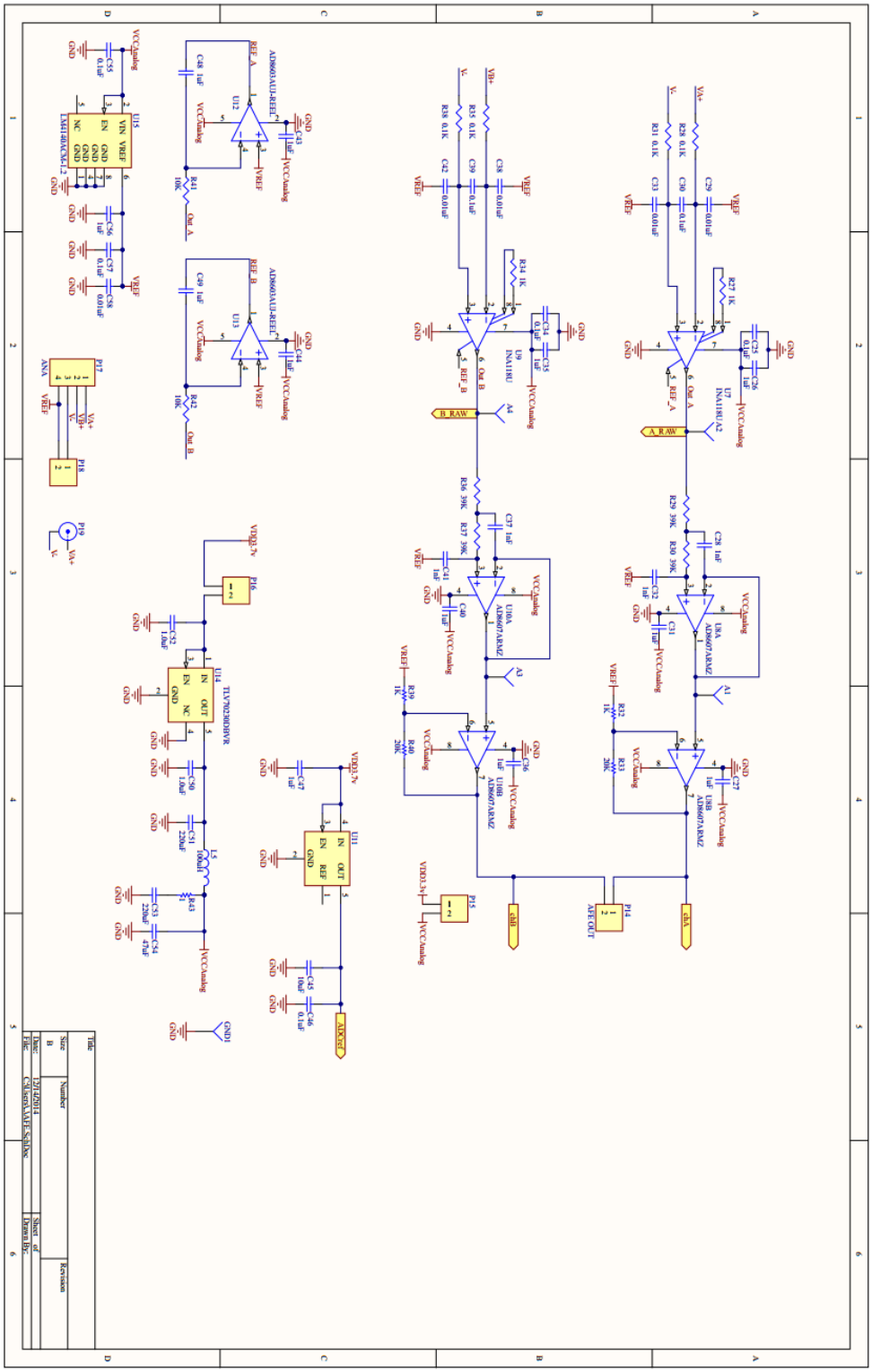
```
    data[0] = 0x0E;
    nRF_reg_write(0x06,data,1);
    //++++++++++++++++++++++++++
    // STATUS : Status Register
    //++++++++++++++++++++++++++
    // OBSERVE_TX : Transmit observe register
    //++++++++++++++++++++++++++
    // RPD : Received Power Detector
    //++++++++++++++++++++++++++
    // RX_ADDR_P0 : Receive address data pipe 0
    data[0] = 0x0F;
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x0A,data,5);
    //++++++++++++++++++++++++++
    // RX_ADDR_P1 : Receive address data pipe 1
    //++++++++++++++++++++++++++
    // RX_ADDR_P2 : Receive address data pipe 2
    //++++++++++++++++++++++++++
    // RX_ADDR_P3 : Receive address data pipe 3
    //++++++++++++++++++++++++++
    // RX_ADDR_P4 : Receive address data pipe 4
    //++++++++++++++++++++++++++
    // RX_ADDR_P5 : Receive address data pipe 5
    //++++++++++++++++++++++++++
    // TX_ADDR : Transmit address
    data[0] = 0x0F;
    data[1] = 0x0E;
    data[2] = 0x0F;
    data[3] = 0x0E;
    data[4] = 0x0F;
    nRF_reg_write(0x10,data,5);
    //++++++++++++++++++++++++++
    // RX_PW_P0 : Number of bytes in RX payload in data pipe 0 ???????????
    data[0] = nRF_packet_len;
    nRF_reg_write(0x11,data,1);
    //++++++++++++++++++++++++++
    // RX_PW_P1 : Number of bytes in RX payload in data pipe 1
    //++++++++++++++++++++++++++
    // RX_PW_P2 : Number of bytes in RX payload in data pipe 2
    //++++++++++++++++++++++++++
    // RX_PW_P3 : Number of bytes in RX payload in data pipe 3
    //++++++++++++++++++++++++++
    // RX_PW_P4 : Number of bytes in RX payload in data pipe 4
    //++++++++++++++++++++++++++
    // RX_PW_P5 : Number of bytes in RX payload in data pipe 5
    //++++++++++++++++++++++++++
    // FIFO_STATUS : FIFO Status Register
    //++++++++++++++++++++++++++
    // DYNPD : Enable dynamic payload length
    data[0] = 0x00;
    nRF_reg_write(0x1C,data,1);
    //++++++++++++++++++++++++++
    // FEATURE : Feature Register
    data[0] = 0x00;
    nRF_reg_write(0x1D,data,1);
    //++++++++++++++++++++++++++
}
//-----------------------------------
```

Appendix C. Headstage Prototype Schematics and PCB Layout

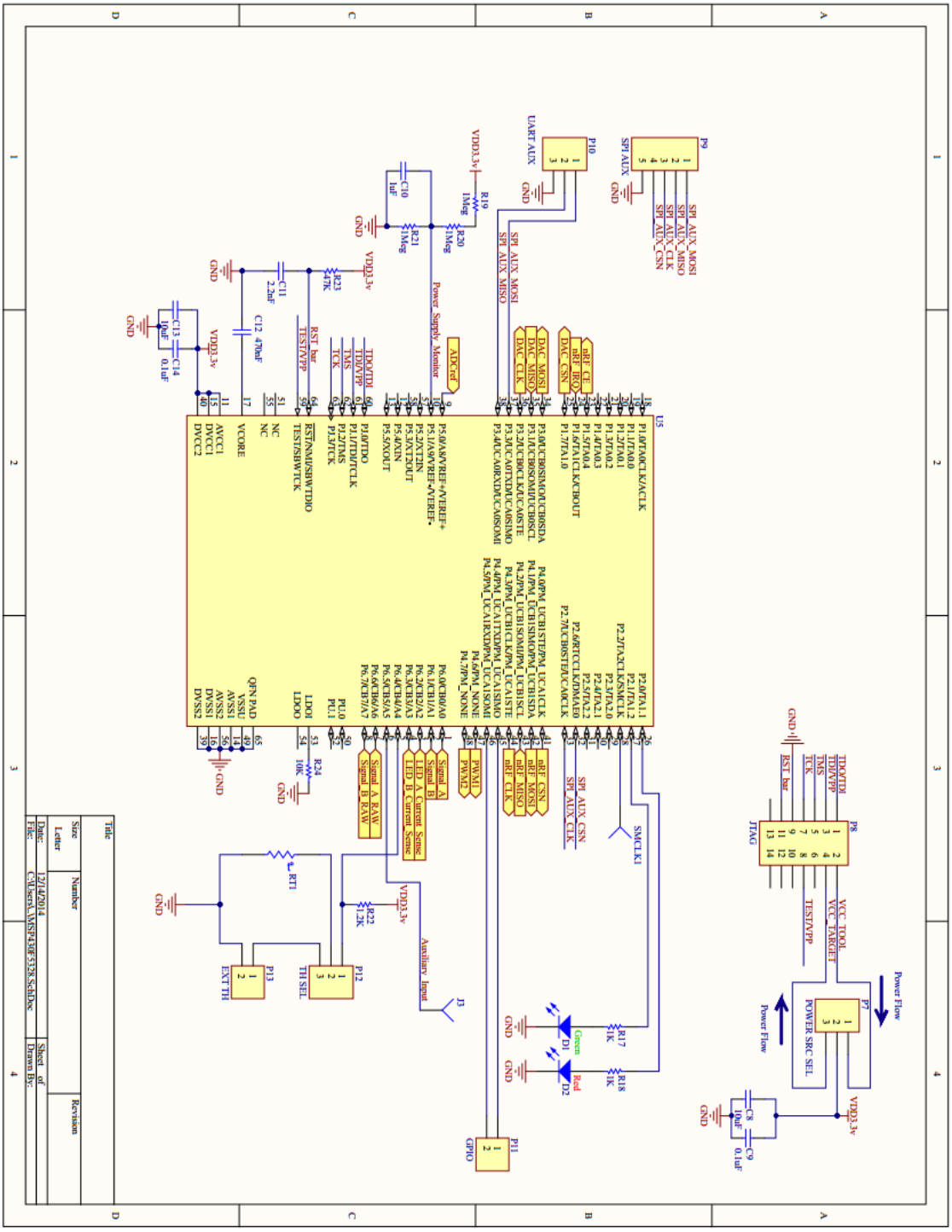
Design Top Module



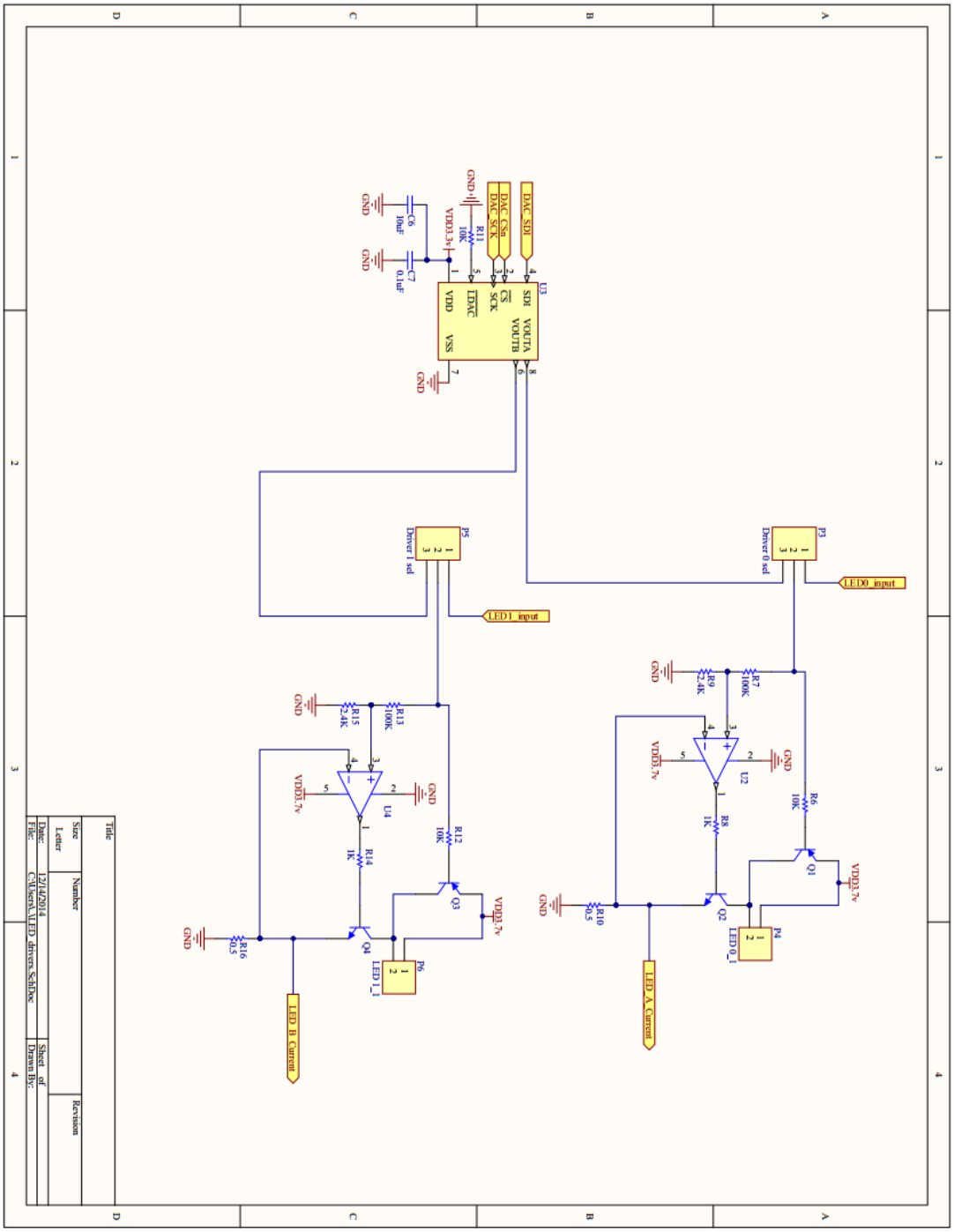
Analog Front End



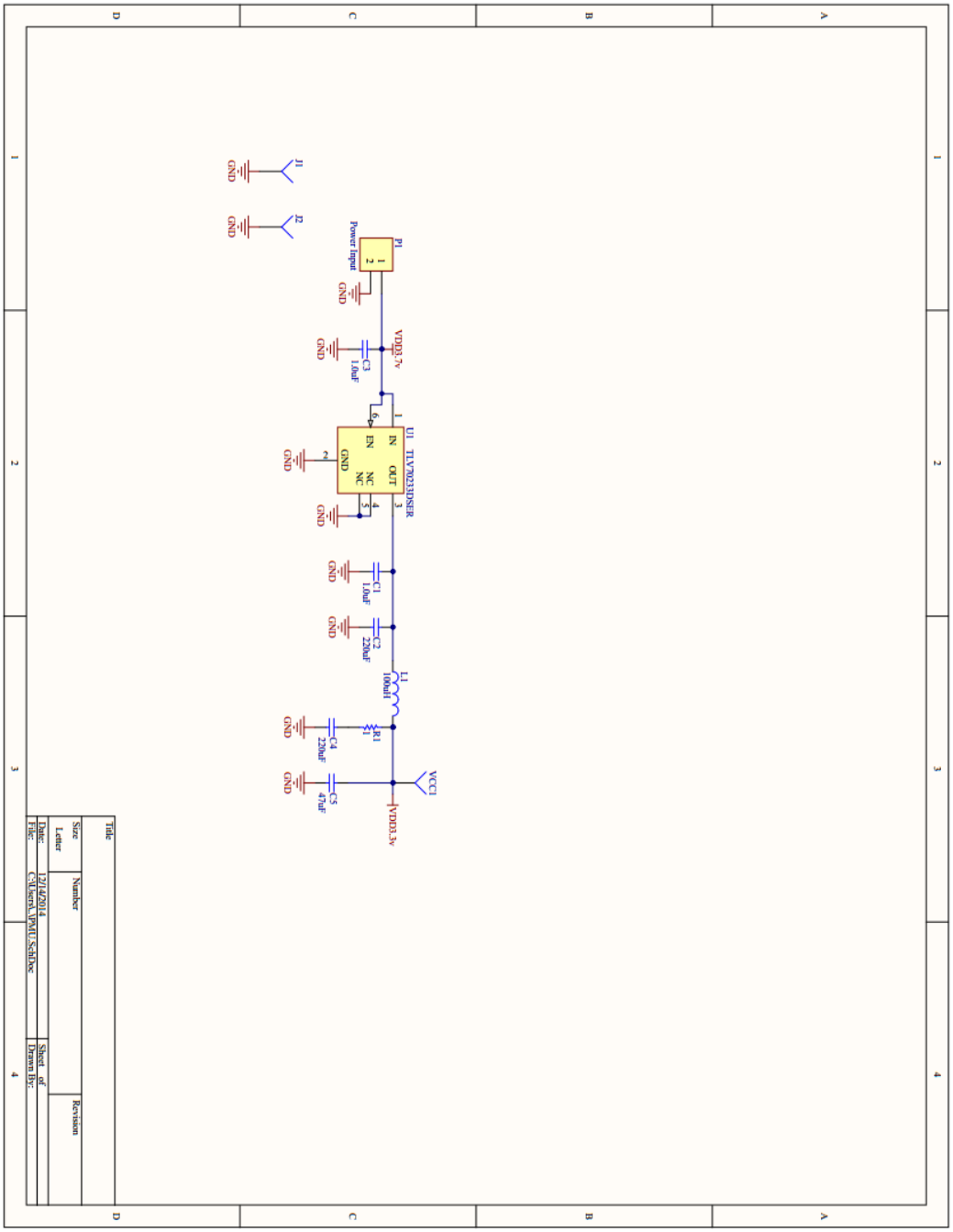
Microcontroller Circuitry



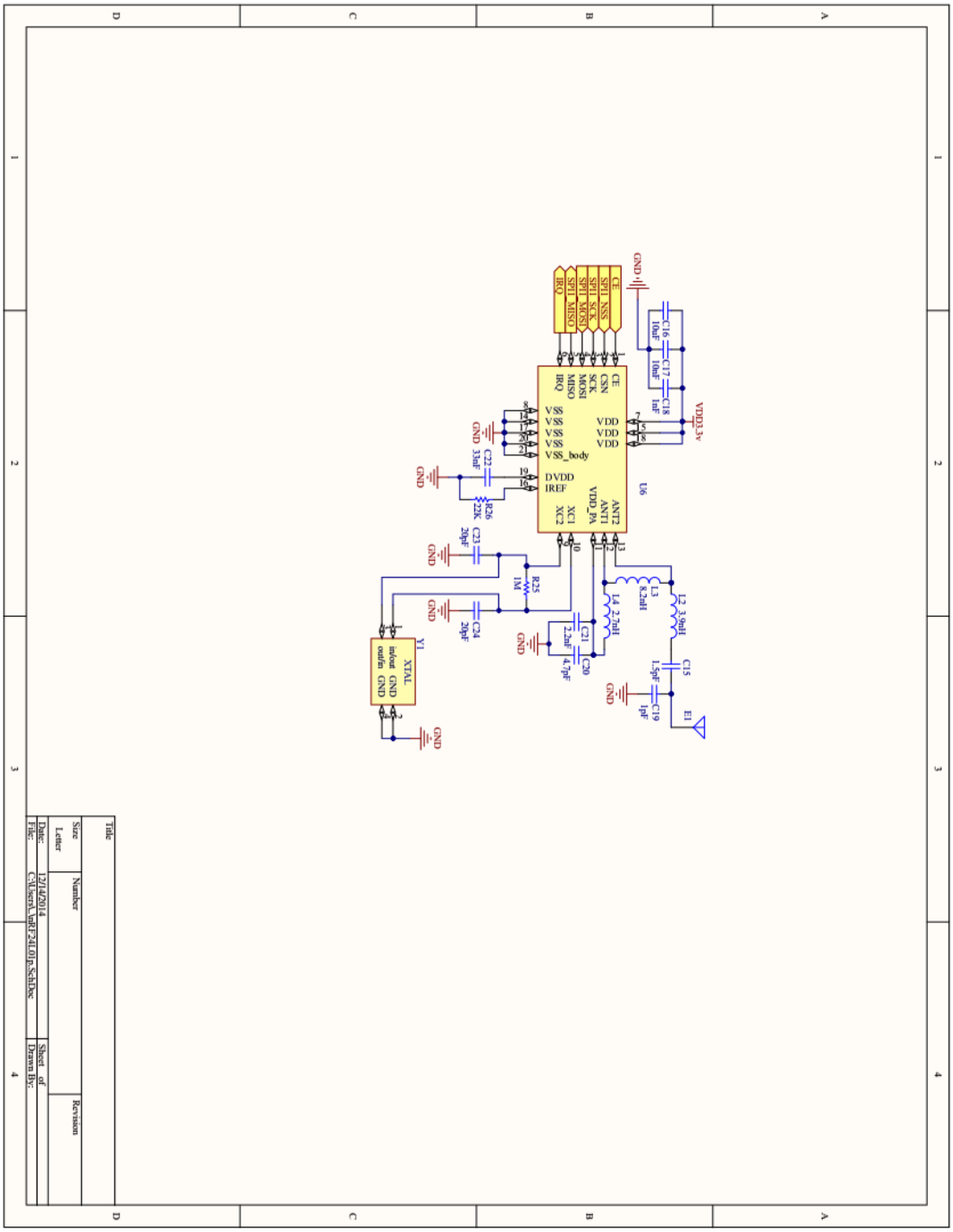
Optical Stimulation Circuitry



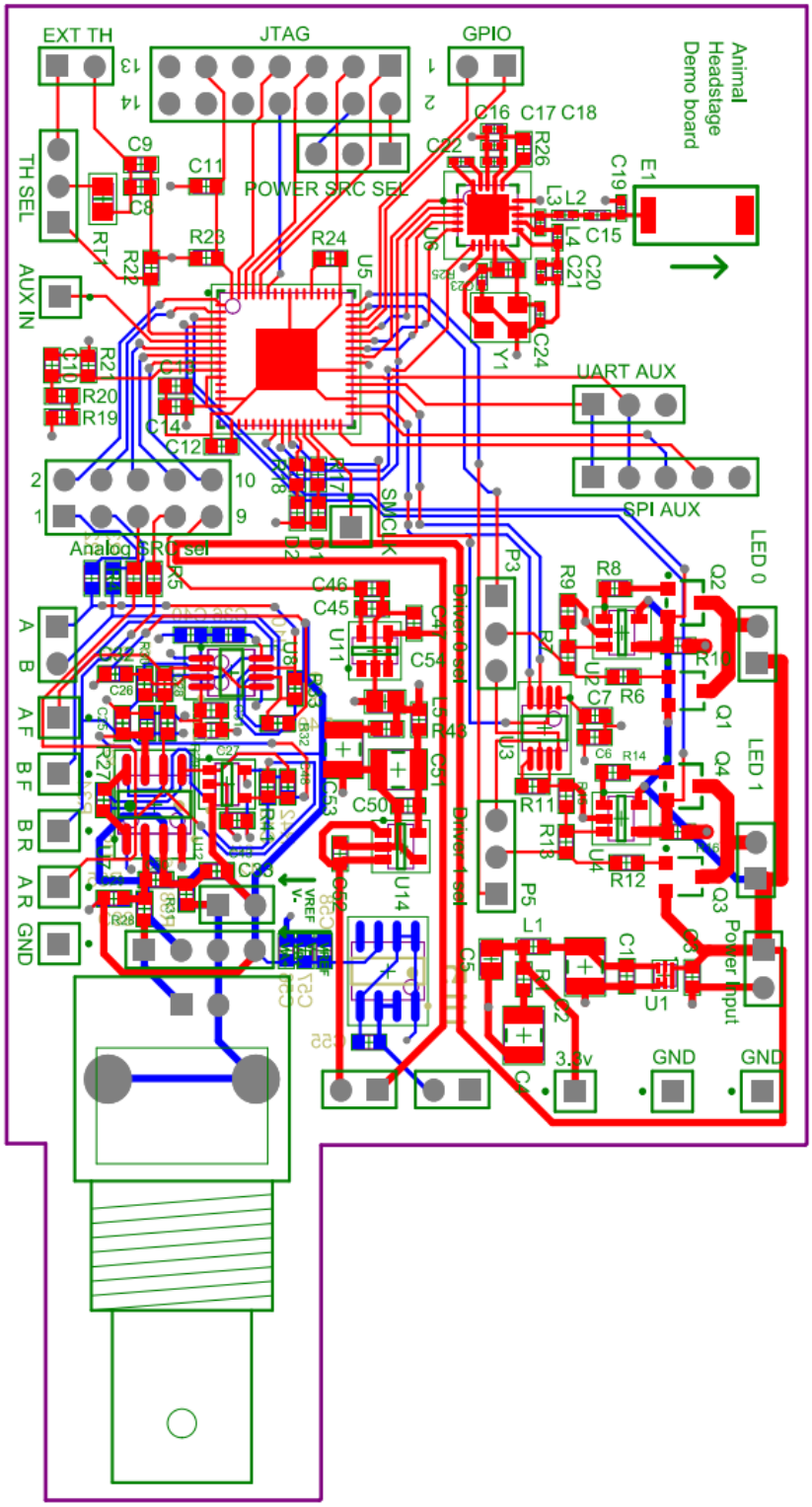
Power Management Unit



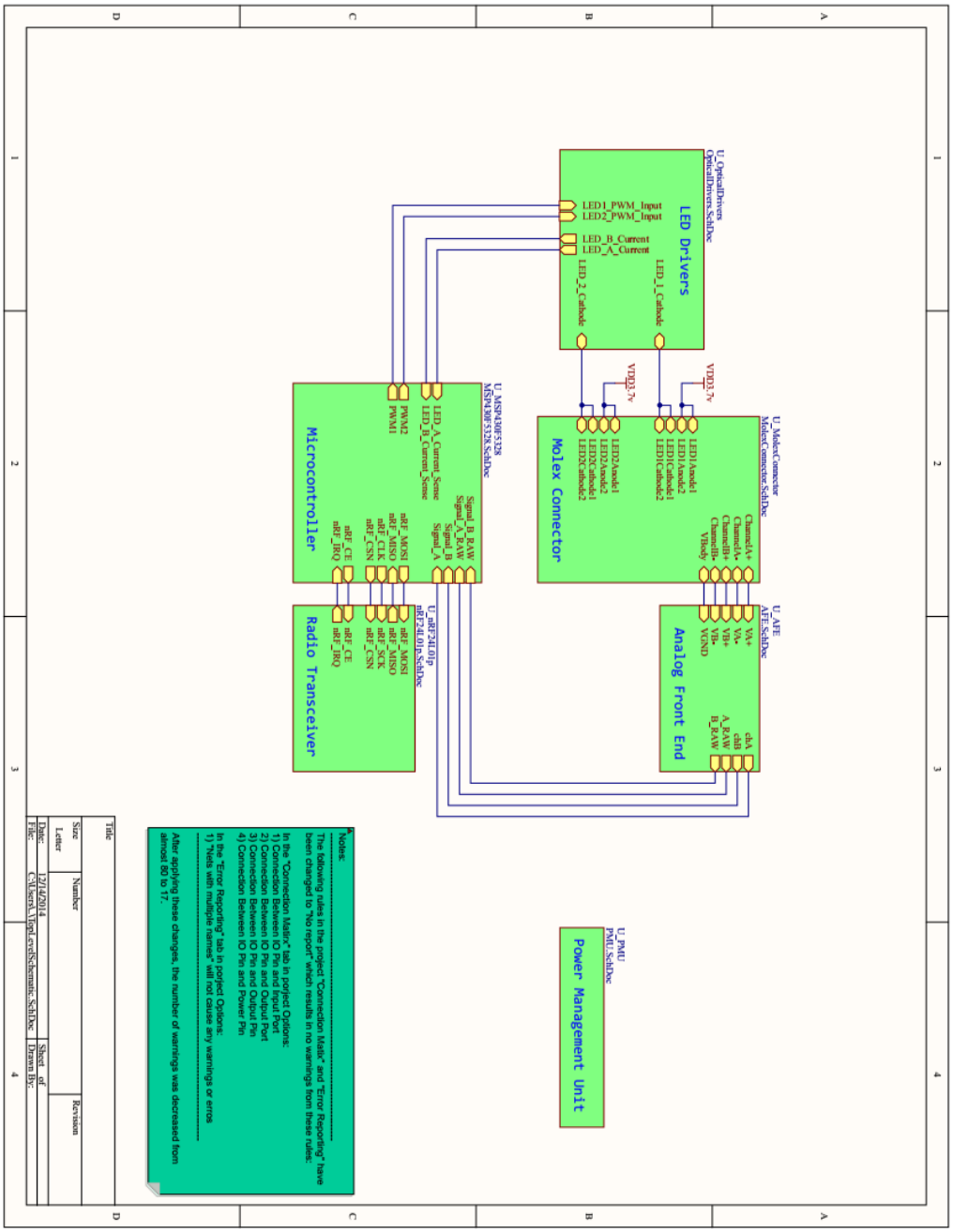
Radio Transceiver



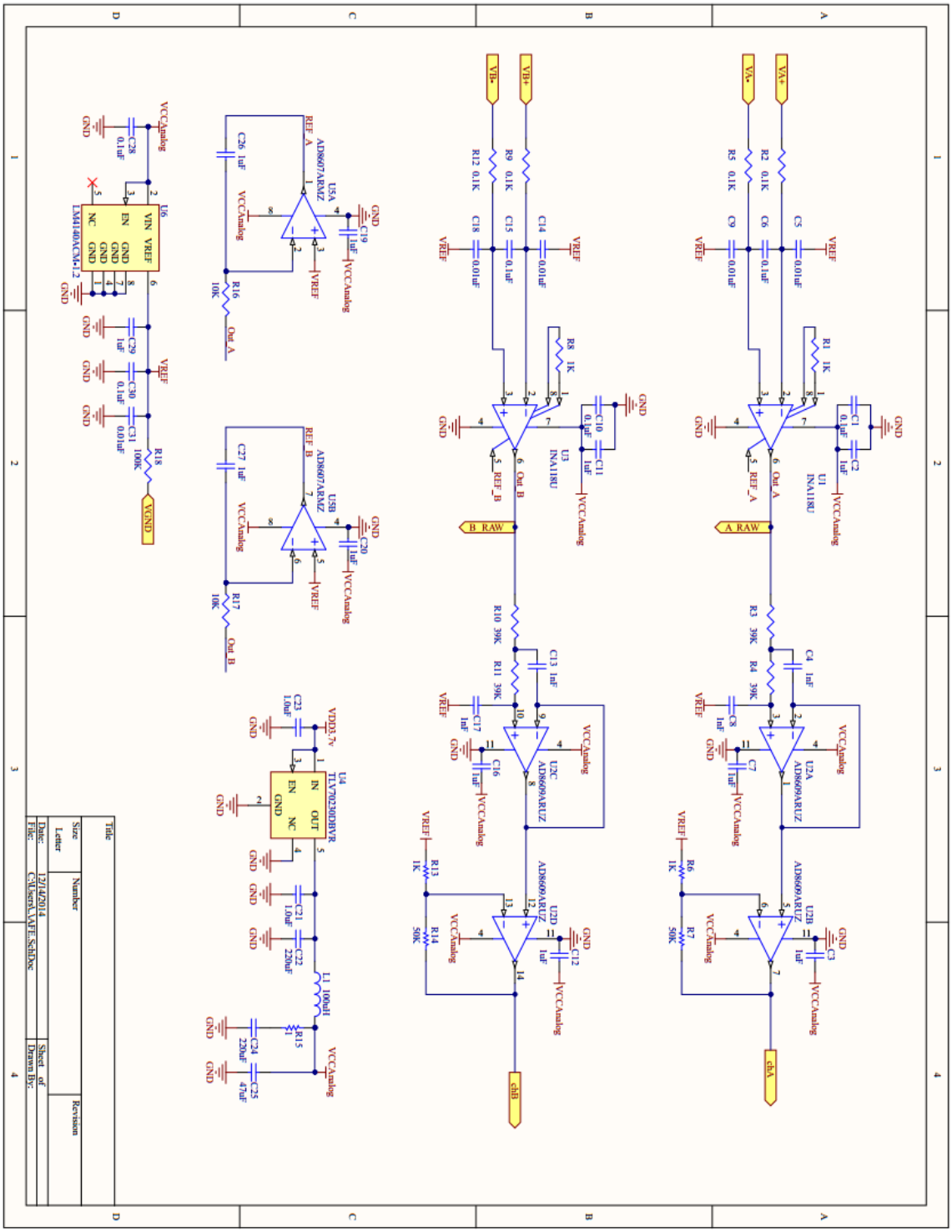
PCB Layout


EXT TH
JTAG
GPIO
Animal Headstage Demo board
TH SEL
AUX IN
POWER SRC SEL
UART AUX
SPI AUX
LED 0
LED 1
Analog PSRC sel
Driver 0 sel
Driver 1 sel
A
B
A F
B F
B R
A R
GND
3.3v
GND
GND
Power Input

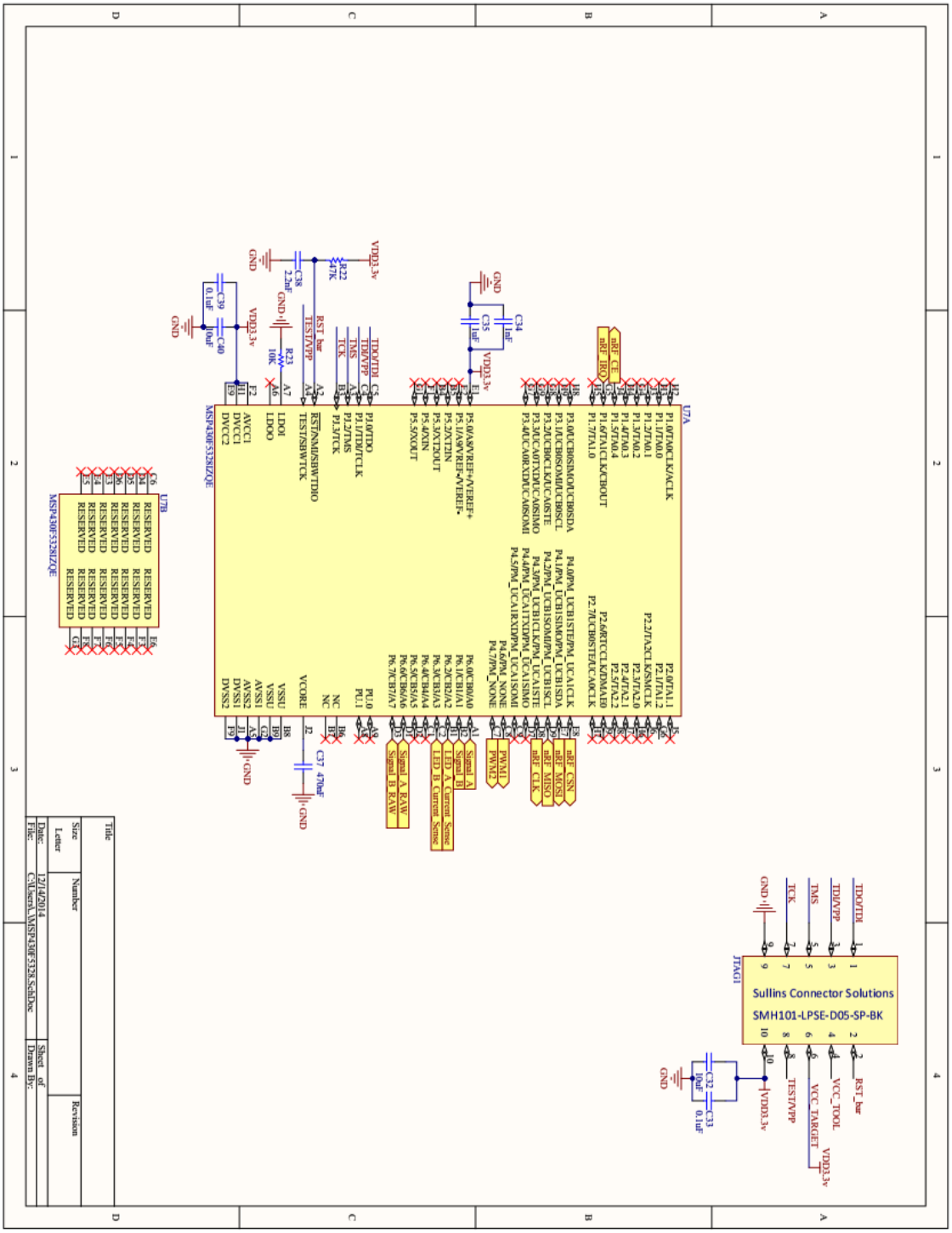
Design Top Module



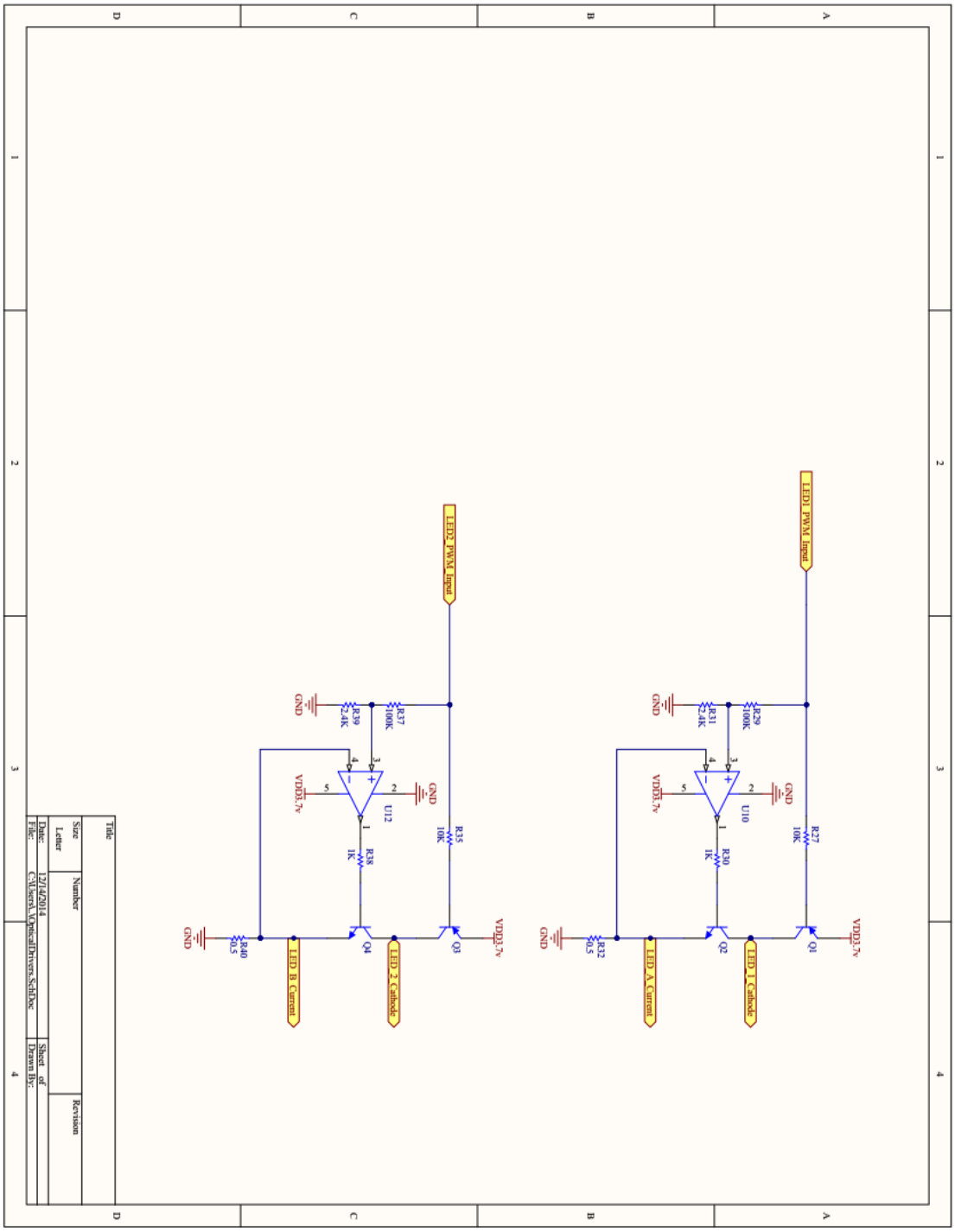
Analog Front End



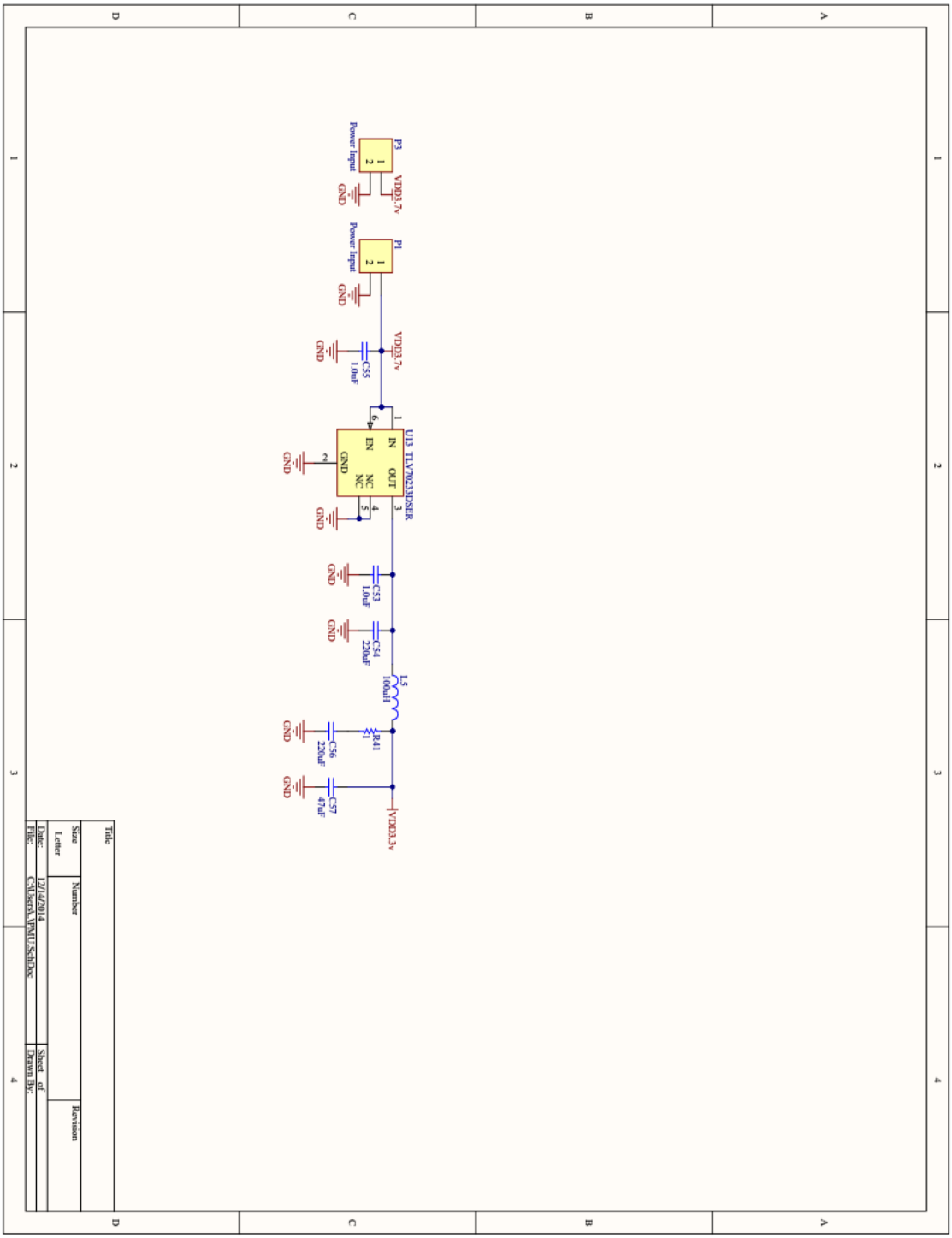
Microcontroller Circuitry



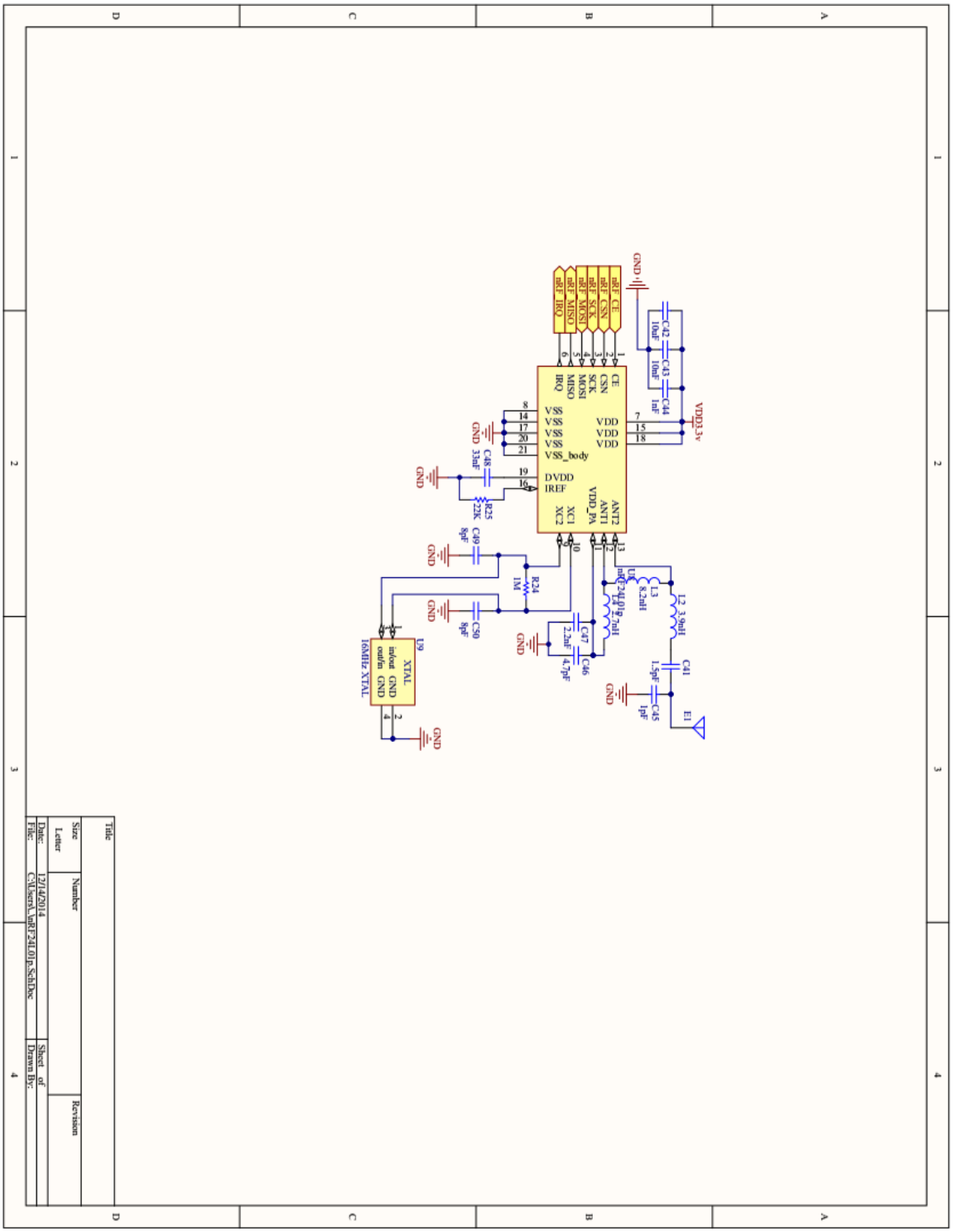
Optical Stimulation Circuitry



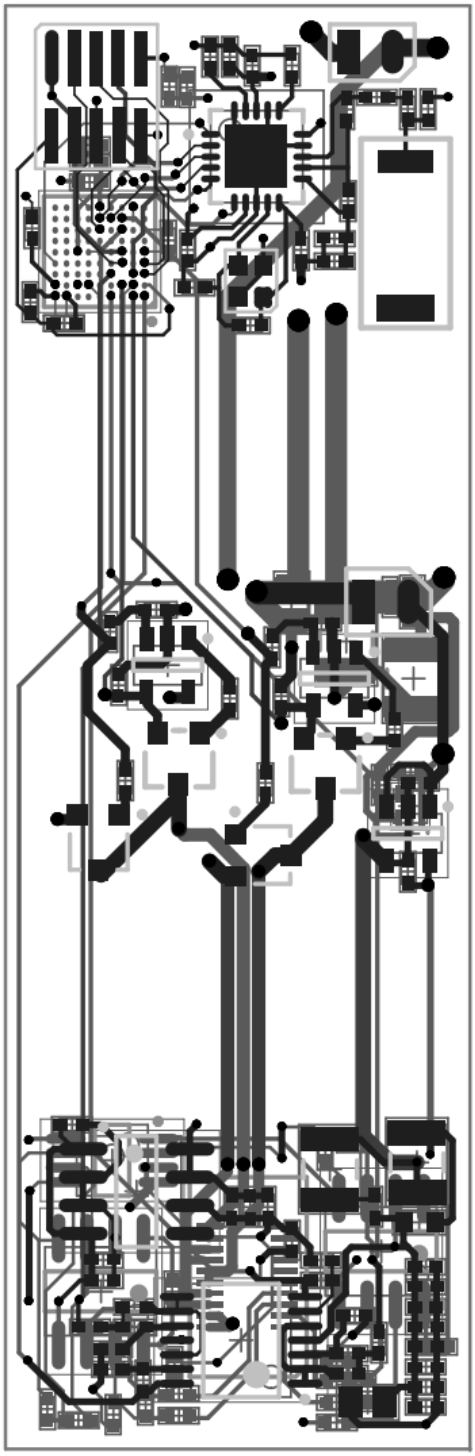
Power Management Unit



Radio Transceiver



PCB Layout



Bill of Materials

Designator	Footprint	Value
C1	0402	0.1uF
C2	0402	1uF
C3	0402	1uF
C4	0402	1nF
C5	0402	0.01uF
C6	0402	0.1uF
C7	0402	1uF
C8	0402	1nF
C9	0402	0.01uF
C10	0402	0.1uF
C11	0402	1uF
C12	0402	1uF
C13	0402	1nF
C14	0402	0.01uF
C15	0402	0.1uF
C16	0402	1uF
C17	0402	1nF
C18	0402	0.01uF
C19	0402	1uF
C20	0402	1uF
C21	0402	1.0uF
C22	C1210	220uF
C23	0402	1.0uF
C24	C1210	220uF
C25	C0805	47uF
C26	0402	1uF
C27	0402	1uF
C28	0402	0.1uF
C29	0402	1uF
C30	0402	0.1uF
C31	0402	0.01uF
C32	0402	10uF
C33	0402	0.1uF
C34	0402	1nF
C35	0402	1uF
C37	0402	470nF
C38	0402	2.2nF
C39	0402	0.1uF
C40	0402	10uF
C41	0402	1.5pF
C42	0402	10uF
C43	0402	10nF
C44	0402	1nF

C45	0402	1pF
C46	0402	4.7pF
C47	0402	2.2nF
C48	0402	33nF
C49	0402	8pF
C50	0402	8pF
C53	0402	1.0uF
C54	C1210	220uF
C55	0402	1.0uF
C56	C1210	220uF
C57	C0805	47uF
E1	ANT-2.45-CHP-B	
JTAG1	SMH101-LPSE-D05-SP-BK	
L1	1608[0603]	100uH
L2	0402	3.9nH
L3	0402	8.2nH
L4	0402	2.7nH
L5	1608[0603]	100uH
P1	TwoSmallPads	
P2	Molex_SlimStack_0537480208	
P3	TwoSmallPads	
Q1	318-08	
Q2	SOT23	
Q3	318-08	
Q4	SOT23	
R1	0402	1K
R2	0402	0.1K
R3	0402	39K
R4	0402	39K
R5	0402	0.1K
R6	0402	1K
R7	0402	50K
R8	0402	1K
R9	0402	0.1K
R10	0402	39K
R11	0402	39K
R12	0402	0.1K
R13	0402	1K
R14	0402	50K
R15	0402	1
R16	0402	10K
R17	0402	10K
R18	0402	100K
R22	0402	47K
R23	0402	10K
R24	0402	1M
R25	0402	22K

R27	0402	10K
R29	0402	100K
R30	0402	1K
R31	0402	2.4K
R32	0402	0.5
R35	0402	10K
R37	0402	100K
R38	0402	1K
R39	0402	2.4K
R40	0402	0.5
R41	0402	1
U1	TI-D8_N	
U2	ADI-RU-14_N	
U3	TI-D8_N	
U4	TI-DBV5_N	
U5	ADI-RM-8_N	
U6	M08A_N	
U7	TI-ZQE80	
U8	QFN50P400X400-20W5M	
U9	8Y-16.000MAAV-T	
U10	TI-DBV5_N	
U12	TI-DBV5_N	
U13	TI-DSE6_V	