Universidade de Lisboa
Faculdade de Ciências
Departamento de Física

Embedded Platform for Electrical Neural Stimulation

Francisco Pargana de Melo

Orientadores

Prof. Dr. Timothy Constandinou

Prof. Ricardo Salvador

Mestrado Integrado em Engenharia Biomédica e Biofísica

Perfil em Engenharia Clínica e Instrumentação Médica

Dissertação

2015

Resumo

Atualmente, o número de tecnologias baseadas em neuro-estimulação está em crescimento, crescimento este que é promovido pelo facto de a neuro-estimulação ser uma área de investigação de elevado interesse devido às várias áreas de possível aplicação, tais como a terapia e tratamento, reabilitação e próteses. Na área da terapia e tratamentos, a possível aplicação da neuro-estimulação está relacionada com a neuro-regulação de órgãos do corpo humano, aplicação que tem sido vista já no campo dos distúrbios do sistema nervoso, tais como a doença de Parkinson e a epilepsia, no tratamento de dor crónica, no controlo do funcionamento cardíaco – no qual se insere uma das tecnologias mais conhecidas, o *pacemaker* – e está ainda a ser investigada para o controlo da libertação de insulina e a absorção renal de sais [1]. Na área da reabilitação, a aplicação da neuro-estimulação tem sido vista em casos de lesões na medula espinal onde a utilização da técnica de estimulação elétrica funcional, ou *FES* de *Funcional Electrical Stimulation*, resultou na recuperação de algumas funções motoras e de controlo de certos órgãos [2]. Relativamente à área das próteses, a neuro-estimulação tem um papel muito importante principalmente no desenvolvimento de próteses funcionais, permitindo que estas próteses não só reajam a informação vinda do sistema de nervoso, realizando os movimentos desejados pelo utilizador, como também forneçam informação ao mesmo, permitindo assim que haja um mecanismo de *feedback* da prótese, aumentando assim a restituição que esta pode dar ao seu utilizador, tanto a nível funcional como emocional. Uma das mais conhecidas próteses que recorrem à neuro-estimulação é o implante coclear que permite uma recuperação parcial da capacidade auditiva recorrendo para isso a um conjunto de microfones no ouvido externo que deteta o som e o transmite à unidade de processamento que por sua vez transforma o som em impulsos elétricos que são direcionados para elétrodos no interior da cóclea e que irão estimular os nervos auditivos [3].

A neuro-estimulação é então um procedimento baseado na estimulação de células excitáveis, como os neurónios, recorrendo para isso à utilização de elétrodos, com o objetivo de iniciar ou inibir um potencial de ação. Esta possibilidade de iniciar um estímulo nervoso através de estímulos externos deve-se ao facto da ativação e propagação de um sinal neural ser um fenómeno eletroquímico. Este fator torna possível o desenvolvimento

de tecnologias que resultem numa maior, ou menor, recetividade da célula a um estímulo através da promoção de alterações do meio em que estão inseridas as células excitáveis ou de propriedades da membrana das mesmas.

O desenvolvimento de tecnologias que recorram à neuro-estimulação está dependente de um estudo profundo dos tipos e estratégias de estimulação de forma a obter a estratégia que seja mais eficaz, segura e eficiente, sendo que esta varia de situação para situação, dependendo de fatores como o local de aplicação e mesmo o resultado que se espera do estímulo. Por estes motivos têm de ser realizados estudos comparativos válidos entre estratégias de estimulação e, para um estudo deste tipo ser valido, os vários estudos devem ser feitos nas mesmas condições com distâncias temporais preferencialmente curtas. Assim sendo, no âmbito da neuro-estimulação recorrendo a estímulos elétricos, criou-se a necessidade de desenvolver sistemas que permitissem uma mais rápida variação dos parâmetros de estimulação comparativamente à montagem experimental clássica. De forma a cumprir estes requisitos, vários sistemas de rápida configuração de parâmetros tem vindo a ser propostos.

O projeto relatado nesta Tese de Mestrado, desenvolvido durante um estágio de seis meses no *Centre for Bio-Inspired Technologies*, *Imperial College London*, apresenta-se então como uma plataforma de neuro-estimulação eléctrica para a realização de estudos comparativos tendo em vista a otimização da estratégia de estimulação, com o objetivo de ser uma versão melhorada dos sistemas já disponíveis. Esta plataforma é composta por três principais componentes: uma interface utilizador-sistema, que permite ao utilizador configurar a estimulação como pretende controlando características como o tipo de onda, a amplitude, a duração, a frequência, entre outros; um microcontrolador e uma placa de estimulação, em que o primeiro controla o segundo de acordo com o que foi configurado pelo utilizador sendo que a placa têm a responsabilidade de gerar e aplicar um estímulo eléctrico. O principal objetivo deste projeto era então desenvolver uma plataforma de neuro-estimulação eléctrica capaz de gerar e aplicar uma estimulação elétrica bipolar com capacidade de equilíbrio de cargas, podendo fazê-lo através de quatro canais de estimulação. Ao mesmo tempo era objetivo que esta fosse pequena, de baixo custo, eficaz, eficiente, de fácil utilização proporcionando um maior leque de possibilidades de configuração comparativamente aos sistemas já desenvolvidos e que pudesse também ser facilmente recriado, alterado e, eventualmente, melhorado.

Os resultados obtidos de testes realizados demonstraram que esta plataforma opera corretamente nos dois principais aspetos do seu funcionamento, nomeadamente a capacidade de gerar uma estimulação de acordo com todos os parâmetros tal como configurados pelo utilizador e a capacidade de cumprir os propósitos de equilíbrio de cargas após estimulação, em todos os tipos de ondas definidos.

Existem no entanto ainda algumas limitações no funcionamento da plataforma. Estas limitações estão relacionadas com a amplitude máxima de estimulação que o sistema é capaz de aplicar, mais especificamente a amplitude máxima do *output* do DAC utilizado e também a amplitude máxima que o amplificador operacional escolhido consegue por no seu *output*; com a existência de algumas imprecisões temporais na aplicação do estímulo, resultantes do tempo de execução de algumas funções por parte do microcontrolador; com o consumo energético e ainda o facto de a ligação entre o computador e o microcontrolador ser feita através de um cabo USB, o que limita a mobilidade que se pode ter durante o trabalho experimental.

Comparativamente a plataformas de estimulação elétrica configuráveis existentes, o sistema aqui desenvolvido apresenta diversas vantagens. Para além de vantagens como baixo custo e facilidade de recriação, esta plataforma tem também um maior número de parâmetros da estimulação que o utilizador pode configurar e também permite uma estimulação através de quatro canais, de três formas diferentes: utilizando apenas um canal, utilizando mais do que um canal ao mesmo tempo ou ainda mais do que um canal de forma sequencial. No entanto, alguns dos sistemas já existentes não apresentam as limitações acima referidas e como tal os desafios futuros desta placa passam por ultrapassar essas limitações.

Palavras-Chave: Estimulação Neuronal, Estudo de Estratégias de Estimulação, Sistema para Configuração de Estimulação Elétrica, MATLAB GUI, Microcontrolador KL26Z, Placa de Estimulação.

Abstract

Nowadays, neurostimulation technologies have grown to reach a wide range of applications including therapy and treatment, rehabilitation and prosthetics and its range continues to grow as it still represents an interesting area of research.

Neurostimulation is based on stimulation of excitable cells, such as nerve cells, through the use of electrodes, with the purpose of achieving initiation or inhibition of an action potential. This interaction is possible due to the electrophysiological base of activation and propagation of a neural signal. This neural signal characteristic makes it possible to use external technologies to promote changes in the nerve cell membrane voltage potential or the environment surrounding it, which can lead to the initiation of a neural signal in the cell or simply to a higher, or lower, receptivity of the cell to a stimulus.

The use of neurostimulation technologies in referred areas and future possibility of use in other applications depends on research developments. An important point of this research is the stimulation strategy, more specifically, the characteristics that a stimulation pulse should have to optimize results towards the intended objective and minimize safety risks.

The present thesis reports a project developed during an internship at the Centre for Bio-Inspired Technologies, Imperial College London which consists in designing and building a full system for the study of stimulation strategies. This full system includes a user interface in a computer, so that the user can choose the stimulus characteristics, such as waveform and amplitude, and define the intended strategy, such as repetition rate and inter-stimulus increasing or decreasing rate; and a microcontroller for control of stimulus application through a front-end stimulation-output circuit, which will be responsible for generation of programmed current-controlled stimulus.

The measurement results verify that the main objectives of this project were accomplished, namely, the capacity to generate a stimulation that meets the parameters as configured by the user and the capacity to carry a charge-balanced stimulation in all the preset waveforms.

However, some limitations were also found related namely with the maximum stimulation amplitude, the small time inaccuracy during stimulation, the power consumption and the fact that connection between the computer and microcontroller is done via USB, limiting the mobility of a experimental procedure using this system.

The system developed here presents some advantages compared to existing systems, such as low cost, easy to build, higher number of parameters that can be configured and can apply stimulation through four channels and do it either with only one, with two or more at the same time or with two or more sequentially. However some of these existing systems do not present some of the limitations mentioned and the challenge on the future of this platform is to overcome these limitations.

Keywords: Neural Stimulation, Study of Stimulation Strategies, Platform for Configurable Electric Stimulation, MATLAB GUI, Microcontroller KL26Z, Front-End Stimulation Board.

Acknowledgements

First of all, I would like to say a great thank you to Professor Timothy Constandinou for this amazing opportunity to do my internship at the Imperial College London and also for all the help and guidance given.

I would also like to thank Dr. Yan Liu and Dr. Song Luan for all the help and guidance given through the whole internship. Their help was of utmost importance to achieve the goal of this internship.

A thank you also to Deren Barsakcioglu, for his help during the development of my work, and to Izabella Wojcicka-Grzesiak, whose patience and help was very important to simplify various issues that arose during my stay.

I would like to thank Professor Ricardo Salvador for all the help and support given during the internship and towards the accomplishment of this thesis.

I would also like to thank the people of the Bio-Inspired group, who were very welcoming, always very friendly and were always available to help me.

A great thank you to my friends Filipa Guerreiro and João Tourais for all the companion and good time during my stay in London, to my friend and roommate Nádia Vilhena for helping to make our home in London a cheerful Portuguese home and especially to my lab partner and roommate Célia Fernandes, whose help, patience and friendship were a great support and very important for me, both at work and at home.

I would also like to express the greatest and most special gratitude to my family for all the support, especially for my parents, Sofia e António. Not only for making possible for me to do another internship outside Portugal, for all the support, both moral and financial, and all the help given, but also for everything that is behind, for the academic and personal path that you gave me which brought me here.

Contents

List of Tables

List of Figures

Abbreviations

AC	**A**lternating **C**urrent
ADC	**A**nalog-to-**D**igital **C**onverter
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuits
CMOS	**C**omplementary **M**etal-**O**xide-**S**emiconductor
DAC	**D**igital-to-**A**nalog **C**onverter
DC	**D**irect **C**urrent
DCU	**D**igital **C**ontrol **U**nit
FSM	**F**inite **S**tate **M**achine
GUI	**G**raphical **U**ser **I**nterface
GUIDE	**G**raphical **U**ser **I**nterface **D**evelopment **E**nvironment
IC	**I**ntegrated **C**ircuit
LDO	**L**ow-**D**rop**O**ut Regulator
MCU	**M**icro**C**ontroller **U**nit
OLED	**O**rganic **L**ight-**E**mitting **D**iode
OpAmp	**Op**erational **Amp**lifier
PCB	**P**rinted **C**ircuit **B**oard
PIC	**P**rogrammable **I**nterface **C**ontroller
RF	**R**adio-**F**requency
SAR	**S**uccessive **A**pproximation **R**egister
SMD	**S**urface-**M**ount **D**evice
SPDT	**S**ingle-**P**ole **D**ouble-**T**hrow
SPI	**S**erial **P**eripheral **I**nterface
SU	**S**timulation **U**nit
THD	**T**otal **H**armonic **D**istortion
USB	**U**niversal **S**erial **B**us

Chapter 1

Introduction and Objectives

Neurostimulation technologies are an area of research with a variety of applications, which range from therapy, treatment and rehabilitation to prostheses. Therapy and treatment applications of these technologies are related to neural regulation of the human body's organs and have been seen in the field of brain disorders – both in movement, such as Parkinson's disease, and psychological disorders, for example epilepsy –, in chronic and untreatable pain, in cardiac control – pacemaker – and are also being studied for the control of insulin release and renal salt absorption [1]. Rehabilitation has been seen in cases of spinal cord disorders, such as Spinal Cord Injury, where some motor and control functions are restored through use of Functional Electrical Stimulation [2]. Prosthetic applications of neurostimulation technologies are related to the use of devices to replace malfunctioning motor or sensory neural functions and in this case neurostimulation is used for the transmission of information from prosthesis to the user's nervous system, as represented by the green line in Figure 1.1. One of the most well-known prosthetic applications of neurostimulation is the cochlear implant, which is composed by a first stage of microphones that detect the sounds, then a processing unit that transforms the sound in electric impulses and transmits them to an electrode array inside the cochlea which is responsible for electric stimulation of auditory nerve fibers [3].

For the aforementioned applications, a careful study of stimulation strategies has to be done so that stimulation can be effective, safe and also energy efficient. Effectiveness varies with location where stimulation is applied and also for the purpose it is intended. For these reasons, valid comparative studies between stimulation strategies have to be

made and for such studies to be valid, a faster capacity of variation of strategy parameters is required compared to the classical methods for study of stimulation. In order to meet those needs, numerous alternative methods have been proposed. The project reported in this Master thesis is presented as an improvement of the existent alternative methods for these comparative studies of stimulation strategies.

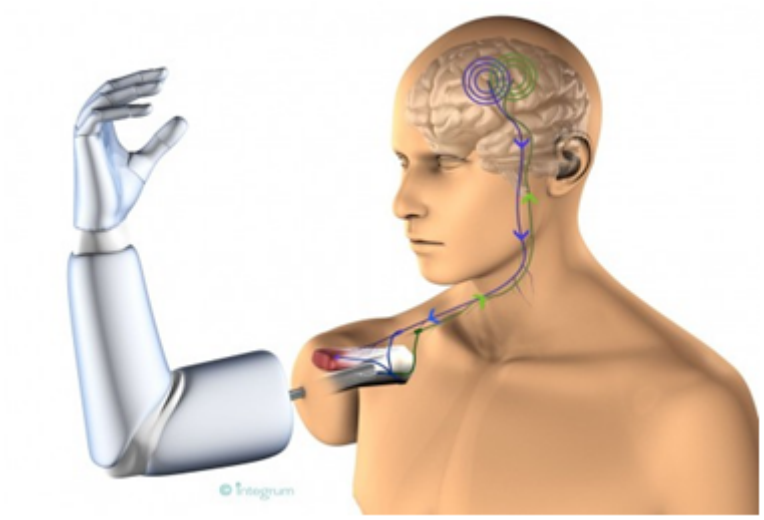

FIGURE 1.1: Prosthesis Control and Feedback through Neural Signal [4].

1.1 Objectives

In this thesis we describe the development of a stimulation system for the study of stimulation strategies that can present itself as an improvement of existing methods that already allow comparative studies of stimulation strategies. In more specific terms, this project is an embedded platform for electrical neural stimulation that allows its user to control the parameters of stimulation, including waveform shape, amplitude, frequency, width and others that will be explained ahead. This system is then capable of generating and applying that stimulation through four different channels. This stimulation system is composed of three main components: a user-system interface, a microcontroller and a front-end circuit board, whose development and operation is described throughout this thesis. The purpose of this project is to obtain a small, low-cost, effective, efficient and easy-to-use stimulation platform that can also be easily recreated, changed and, eventually, improved.

1.2 Thesis Structure

The thesis here presented is divided in six main chapters. The first main chapter is Chapter 2, "Fundamentals of Neural Stimulation", where the principles behind neural stimulation are explained, namely the anatomy and physiology that lie under the functioning of the nervous system and also the mechanisms behind an external electrical stimulation.

In Chapter 3, "State-of-the-Art of User Configurable Neural Stimulation Technologies", a state of art of user-controllable systems for electrical neural stimulation is made, including a description of main advantages brought to this field of investigation and also limitations attached to it.

Chapter 4, "Software Development", is dedicated to the development and operation of the software part of the stimulation system. This chapter is divided in two sections: the first section focuses on the user-system interface, where it is described the options and parameters that user is able to configure, the operation of the interface behind the choices of the user and the interface side of interface-microcontroller communication; the second section focuses on the microcontroller side of interface-microcontroller communication and its operation.

In Chapter 5, "Hardware Development", the development and operation of hardware part of the system, more specifically the 'front-end' circuit board, is described. This includes the requirements, schematics and simulations behind the development up to the schematics and sketches of resulting final board.

Chapter 6, "Overall Stimulation System Operation", presents a discussion on the embedded platform created, with a specific focus on the results obtained using this system to apply different types of stimulation with different waveform parameters through different channels and on the system charge-balancing capacity.

Lastly, Chapter 7, "Conclusion and Future Work", presents important conclusions on this project and also perspectives on future work. Following the last chapter are appendices related to Chapter 4, namely the code for the user-system interface, and to Chapter 5, namely the code for the main function of the microcontroller.

Chapter 2

Fundamentals of Neural Stimulation

The major principle of neurostimulation technologies is the activation of excitable neural tissue through use of electrodes. This type of activation of neural tissue is possible due to the electrochemical nature of neural communications in human body. Neural communication via electric pulses is possible due to a series of unique anatomic and physiological characteristics of cells that compose the nervous system, and particularly the neural cell. Thanks to these characteristics, these cells have the ability to receive, conduct and send electrical-based signals throughout the nervous system. These electrical-based signals are called action potentials, i.e., rapid oscillations of the transmembrane potential resulting from ionic currents. The following chapters thoroughly explain these fundamentals of neural stimulation, more particularly, anatomy and physiology behind neural communication, specifically the neural cell and mechanisms of action potential generation, and the mechanisms behind electrical neural stimulation.

2.1 Neuroanatomy and Neurophysiology

As stated previously, the basis of neurostimulation is the electrical nature of neural communication in the nervous system, whose base by its turn is the neural cell. In the following chapters an overview of both these bases is done, more specifically, the

neural cell structure and proprieties and the mechanisms behind the generation of action potentials.

2.1.1 Neural Cell

The neural cell, also known as neuron, is, as it was mentioned previously, the base of the neural communication and it consists of an electrically excitable cell composed by dendrites, a cell body, an axon and terminal branches, as it is presented in Figure 2.1. For example, the normal path of an electrical signal on the neuron starts by the reception from other cells via the dendrites which is then transmitted to cell body, the metabolic and processing centre, which then sends it through the axon – a fibre that can be surrounded by myelin sheaths with intermittent unmyelinated areas, called nodes of Ranvier, whose conformation allows a higher speed of propagation of electric signal – to the terminal branches where information is passed to other cell via a process called synapse. This direction of propagation is called orthodromic propagation. But signals can also propagate antidromically, i.e., from axon to dendrite. A synapse is an electrical or chemical process, depending on the way the signal is transmitted to the following cells, in which the electrical signal is transmitted from the presynaptic cell to the postsynaptic cell. In an electrical synapse, the electric signal passes from presynaptic to postsynaptic cell through channels that connect the cytoplasm from both cells. These channels are called gap junctions. In a chemical synapse, when the electrical signal coming from the axon reaches the terminal branches it causes a release of neurotransmitters which leave the cell to bond to receivers in the postsynaptic cells. The latter triggers a reaction in ion channels in the post-synaptic membrane which can lead to changes in its membrane potential. [5]

A very important characteristic of the neural cell is the presence on its cell membrane of transmembrane proteins, called ion channels and ion pumps. These proteins are crucial on the cell excitation process since they selectively allow ions to cross the cell membrane, since the phospholipidic layer is impermeable to ions, providing also a control of ion concentration gradient across the cell through ion permeability variations and also active transportation. This excitation process is behind electrical signal generation and propagation as it will be explained in the following chapter. [5]

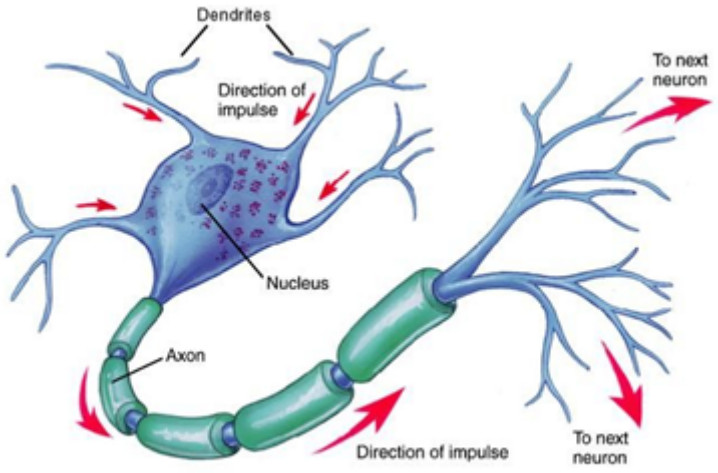

FIGURE 2.1: Representation of Neural Cell [6].

2.1.2 Action Potential Generation

In the absence of external changes, the neuron is found in a resting state where there is a difference between extracellular and intracellular ion concentrations. This difference is caused by a much higher presence of Sodium cathions Na^+ and Chloride anions Cl^- on the outside of the cell than on the inside where there is a much higher presence of Potassium cathions K^+ and negatively charged particles such as proteins or phosphate compounds. This difference of concentrations is mostly caused by sodium-potassium pumps existent in the neurons membrane since they promote, through active transport, the flux of potassium ions into the cell and sodium ions outside the cell. Besides sodium-potassium pumps, selective permeability of cell membrane also plays an important role in different concentrations since the flux of certain substances is done through active and passive ionic channels. [7]

Due to these different ionic concentrations inside and outside the cell membrane, the inside is a positively-charged medium while the outside is a negatively-charged medium, which creates a voltage potential across the membrane, called resting potential. This resting potential is then a negative value, usually of about -70 mV. When a stimulus is applied to the cell, a variation of this potential occurs originating a local potential on the membrane. If this potential reaches a certain threshold, for example -55 mV – see Figure 2.3 – a sharp increase of the membrane potential occurs, followed by a rapid descrease to restore the resting potential. This is called an action potential and it obeys

to an "all-or-none" principle, which defines that it only occurs if necessary conditions are verified, i.e., only if stimulus intensity is higher than referred threshold. In this situation, other membrane potential variations occur in adjacent places of the membrane, leading to a sequence of potential changes throughout the membrane and, consequently, throughout the neuron. Otherwise, if stimulus does not reach the threshold, there is small variation of the cell membrane potential and no action potential is generated. In this case, there is little or no propagation throughout the membrane. [7]

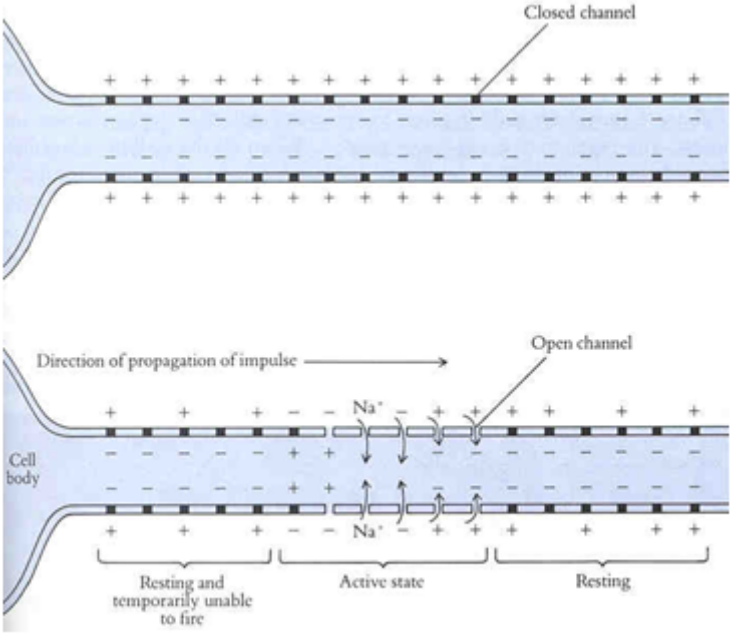

FIGURE 2.2: Action Potential Propagation throughout a Nerve Cell [8].

Despite being a consequence of a stimulus, action potentials are generated by the cell membrane, whose voltage-gated Sodium channels open, allowing influx of Sodium ions, thus increasing the membrane potential. This phase is called membrane depolarization. Voltage-gated channels are channels whose opening and closing for its specific ions depends on the membrane voltage across near the channel. When the membrane potential reaches a certain maximum positive value, the Sodium channels get shut off slowly, reducing its flux. On the other hand, the membrane becomes more permeable to Potassium ions that initially also flow to the inside of the cell and make it even more concentrated there, which then leads to outside diffusion according to the concentration gradient. This diffusion to the outside of the cell will make the intracellular medium negatively-charged and the extracellular medium positively-charged again, in an attempt

to restore the membrane potential – a process called repolarization. After repolarization, Sodium pumps lead Sodium ions back to the outside of the cell, which makes the intracellular medium slightly more negative than its resting state. This slightly negative phase is called refractory period and allows the potassium ions to return to its original place, the inside of the cell, through active transport and diffusion, which restores the membrane to its resting potential. During the refractory period, also called hyperpolarization phase, Sodium and Potassium channels do not react even if a new stimulus is applied. This is important for the unidirectionality of propagation of action potentials in the neuron. [7] An illustration of the previously described events is presented in Figure 2.3. Besides having a role in the membrane resting state potential, the Chloride anion diffusion effect is negligible when an action potential occurs [7].

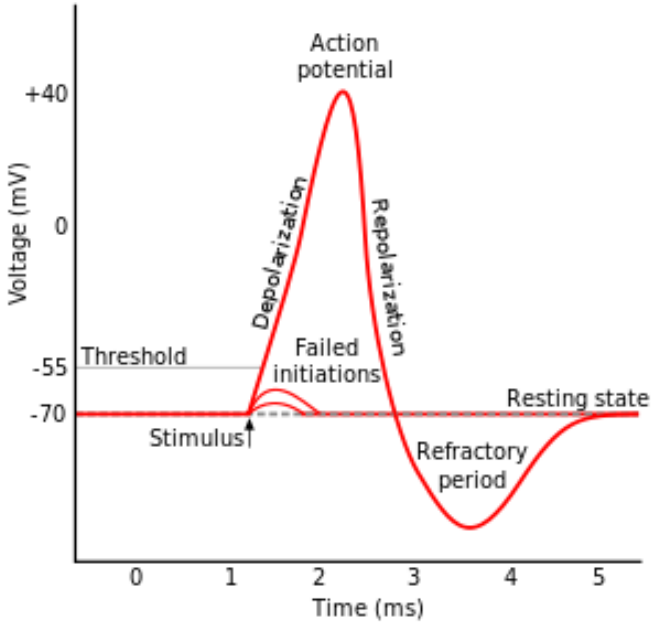

FIGURE 2.3: Variation of Membrane Potential during an Action Potential [9].

The basic mechanism behind neuronal activity was first identified by Hodgkin and Huxley in 1952. They developed a mathematical model for action potential initiation and propagation along the nerve which was based on a circuit model of cell membrane. For this circuit, every component of the cell membrane is represented by an electric component: the dual phospholipid layer is represented by a capacitor C_m, the voltage-dependent ion channels as conductances g_n and the electrochemical gradients as voltages sources E_n, where n represents each of the ions to which the membrane is permeable

to. The circuit is represented in Figure 2.4 and V_m represents the voltage across the cell membrane [7].

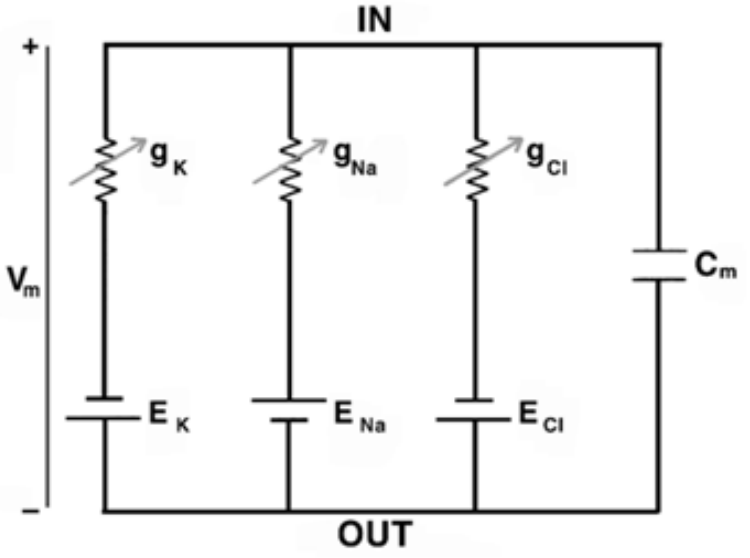

FIGURE 2.4: Circuit Model of Cell Membrane [7].

The Hodgkin and Huxley model then describes the total membrane current I_m as the sum of the ionic currents I_{ion} and capacitive current I_C. This results in Equation 2.1.

$$I_m(t) = I_{ion}(t) + I_C(t) \tag{2.1}$$

In this equation, I_{ion} is given by the sum of the currents that pass through each of the ion channels, as it is on Equation 2.2. In this equation, Na and K represent sodium and potassium channels, respectively, and L is the leakage current which represents the less significant ion channels. Each of ionic currents is time dependent and its value is obtained from the conductance of the ionic channel and the potential difference between membrane potential V_m and Nernst Potential E_n for each ion, as in Equation 2.3. The Nernst Potential is the potential difference across the membrane at equilibrium. Equilibrium is the state where the rate of movement due to diffusion of the ion n in one direction, resulting from ionic concentration differences between the inside and outside of the membrane, is equal to its rate of movement in the opposite direction due to electrical field, which results from the transmembrane potential created by the accumulation of charged ions in the membrane. The capacitive current I_C represents the change of charge inside the cell and is given by Equation 2.4.

$$I_{ion} = \sum I_n = I_{Na} + I_K + I_L \tag{2.2}$$

$$I_n(t) = g_n(t, V_m) \times (V_m(t) - E_n) \tag{2.3}$$

$$I_C(t) = C_m \frac{dV_m(t)}{dt} \tag{2.4}$$

In ionic current equation, Equation 2.3, conductance of ionic channel is given by Equation 2.5, where $\overline{g_n}$ is the maximum conductance of respective ionic channel, and δ and ρ are the probability of existing channel subunits being open. In these, q and k are the number of correspondent subunits existent in the ion channel. For its turn, the probability of subunits being open is given by Equation 2.6, where α_δ are the opening rates and β_δ are the closing rates of the subunits and are both time- and voltage-dependent, δ_0 is the probability of open channels in $t = 0$, and v_m is the reduced membrane potential, which is the difference between the membrane potential V_m and the membrane resting potential V_r.

$$g_n(t, V_m) = \overline{g_n} * \delta^q(t, V_m) * \rho^k(t, V_m) \tag{2.5}$$

$$\frac{d\delta(t, v_m)}{dt} = \alpha_\delta(v_m) * (1 - \delta) - \beta_\delta(v_m) * n \tag{2.6}$$

In membrane-ionic dynamics, temperature also plays an important role, namely in membrane kinetics and ionic channel conductance, so a thermic coefficient $\emptyset$ can be included in both Equation 2.3 and 2.6 to represent acceleration in membrane kinetics. This thermic coefficient is multiplied by the referred equations and is given by Equation 2.7. In this equation, Q_{10} is a constant that represents the increase in permeability and kinetics of the membrane when temperature raises 10^oC and T_0 a reference temperature. [10]

$$\emptyset = Q_{10}^{\left(\frac{(T-T_0)}{10}\right)} \tag{2.7}$$

2.2 External Electrical Neuron Stimulation

As it was mentioned in the previous chapter, nerve stimulation is possible because neural communications are based on electrochemical changes or more precisely, as explained, ionic currents. An intuitive way of inducing nerve stimulation is then to apply an electric current to the excitable nerve tissue. This can be achieved through the placement of at least two electrodes, a positive anode and a negative cathode, in the neural tissue and promote the passage of electrical current from one electrode to the other. These electrodes can also be called sink and drain electrodes since their function is to inject current and then remove it from the tissue. This is known as a bipolar montage. When current is applied to an electrode, free electrons of the electrodes' metallic conductor reach the electrode-biological tissue interface where there is a transduction of charge carriers, from electrons to ions, which allows the current to continue its path [11]. A further description of this phenomenon will be made in the following subsection.

2.2.1 Electrical Stimulation Principles

When an electrical stimulation is applied through the use of electrodes there is one interface to be considered besides the aforementioned electrode-tissue interface - the electrode-electrolyte interface. The two interfaces are represented in Figure 2.5 and Figure 2.6.

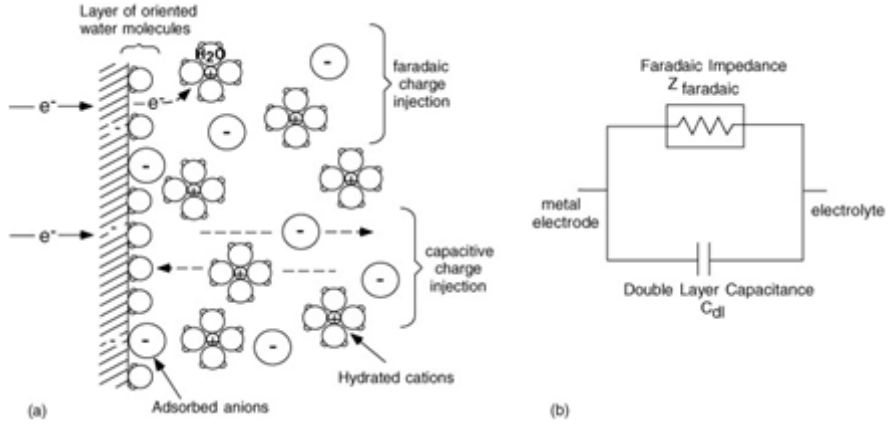

FIGURE 2.5: Electrode-Electrolyte Interface Representation (a) and Circuit Model (b) [12].

On the representation of electrode-electrolyte interface circuit model, Figure 2.5.b., we can see that the interface consists of a resistor $R_{faradaic}$, which represents the Faradaic

resistance, in parallel with a capacitor C_{dl}, which by its turn represents the Helmholtz double-layer capacitor effect at the interface. A Helmholtz double-layer, or to simplify double-layer, is a structure composed of charged particles, and ions attracted to these particles, adsorbed to a structure of an metal object when it is exposed to a fluid – Figure 2.5.a. Due to this capacitor effect, one can infer that the impedance of electrode is frequency dependent. Also contributing to a difficult accurate modelling of this interface is the fact that both C_{dl} and $R_{faradaic}$ are nonlinear time-varying components. [13] A circuit model of a bipolar electrode montage is represented in Figure 2.6, where the resistor R_S is the access resistance and can represent either the electrolyte or the tissue resistance.

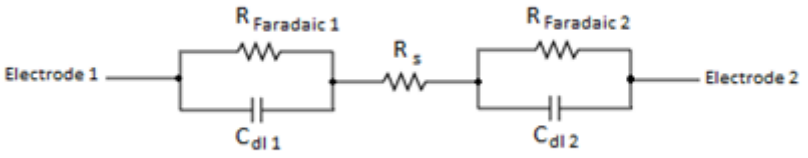

FIGURE 2.6: Electrode-Electrolyte and Tissue Interface Circuit Model in a Bipolar Montage [11].

Half-cell potential for the electrode is also a factor to be considered. Half-cell potential results from chemical reactions in the double-layer that momentarily pumps electric charged particles between the electrode and electrolyte creating a potential difference. The half-cell is simply a structure composed of a conductive electrode and a conductive electrolyte surrounding it where a naturally occurring double-layer separates them. [14] However, in the case of a dipole electrode montage, the half-cell potential of both electrodes can be omitted from the model because it is assumed that they are made of same material and so they cancel each other. [11]

The importance of this model for stimulation designs is that it allows studying each mechanism and reaction behind the application of an electric stimulus through an electrode. There are two main mechanisms that allow the passage of current between two different media: the Faradaic and the non-Faradaic mechanism.

A non-Faradaic mechanism, also known as capacitive charging process, is related to a redistribution of charged chemical species in the electrolyte and the referred Helmholtz double-layer capacitor effect charging and discharging. In this mechanism, charge is

applied on each electrode, creating a capacitor-like effect where the tissue acts as a dielectric layer. This way, charge on the electrode attracts, or repulses, ions from the tissue thus producing stimulating pulses of ionic current. In a perfect non-Faradaic process, there is no charge crossing the electrode-electrolyte interface and a situation close to this is possible as long as charge density is maintained below the electrode's limit. This ideal situation is represented in the circuit model by an $R_{faradaic}$ with infinite value. In this mechanism, it is possible to reverse the charge distribution by reversing the direction of applied current. [11]

In a Faradaic mechanism, also known as electrochemical reaction process or oxidation-reduction reaction process, an injection of charge occurs from electrode to electrolyte and in the opposite direction. An oxidation reaction is verified at the positive electrode, resulting in removal of electrons, and at the negative electrode a reduction reaction occurs, which acquires electrons. It can then be divided in two types of reactions: reversible and irreversible. The reversibility of a Faradaic process is dependent on relative rate of kinetics and mass transport. For a Faradaic process to be reversible, the kinetic rate, which is the electrons transport at the interface, has to be higher than the mass transport rate. In this situation, redox products formed at the interface are stored near the electrode instead of diffusing away and a reversion of stimulus current direction allows the products to return to its original forms. In an irreversible Faradaic process, a smaller kinetic rate is verified and so the redox products are able to diffuse away instead of being stored near the electrode and not even the inversion of current direction is able to reverse the products into their original forms. [13] An irreversible Faradaic is then a process to be avoided since it leads to electrochemical changes in the interface, such as electrode dissolution and gassing or pH variation of electrolyte, which can generate toxic chemical substances to the tissue. [11]

To apply a safe stimulation, a charge balancing phase after the stimulating phase has to exist to avoid irreversible Faradaic charge transfer. This charge balancing phase should be the application of a current with the opposite direction of the stimulating current in order to recover all the charge initially injected, as soon as possible to avoid diffusion of the redox products. [11] However, in most electric stimulation systems an interphase delay between stimulating and charge balancing is used so that stimulation effect can be maximized. [15]. Figure 2.7 exemplifies the referred stimulation by showing a classic square pulse stimuli followed by an interphasic delay and then a charge balancing phase.

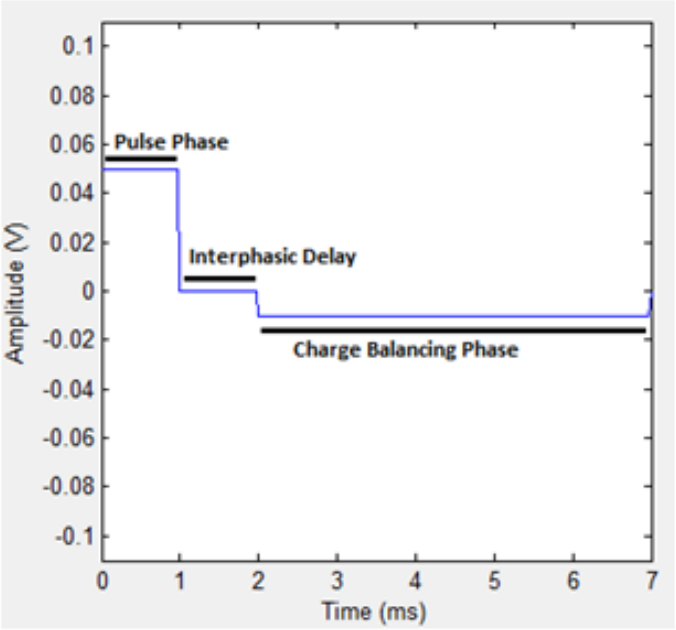

FIGURE 2.7: Biphasic Square Pulse Stimulation.

For the charge balancing phase to be precise so that safety is maximized, the amount of charge in stimulation and charge in this balancing should be the same. However, if amplitude in balancing phase is of the same amplitude as the pulse, a new stimulation or inhibition can occur. To solve this, the amplitude and duration are changed so that the amplitude can be reduced but the amount of charge stays the same. This decrease in amplitude therefore means an increase on duration. This increase and decrease are usually calculated through a ratio, called biphasic ratio, applied to the referred parameters of original pulse. Equation 2.8 and 2.9 present the calculation of charge balancing amplitude A_{cb} and duration T_{cb}, respectively, through the use of biphasic ratio B.

$$A_{cb} = \frac{A_p}{B} \tag{2.8}$$

$$T_{cb} = T_p * B \tag{2.9}$$

A_p and T_p respectively represent the amplitude and duration of the original stimulating pulse.

2.2.2 Electrical Stimulation Application Techniques

The principles described in the previous section are the base of electrical stimulation which essentially consists of applying a potential gradient across the neuron, using electrodes, which will create an intracellular ionic current flow and depolarization or hyperpolarization of cell membrane that can result in neural stimulation or inhibition, respectively.

Application of these potential gradients in the neural tissue is currently achieved through three electrical ways, either by Voltage-Controlled, Current-Controlled or Charge-Controlled stimulators. Voltage-Controlled stimulators' operation is based on application of a voltage between the electrodes. This type of stimulators is a more simple way of applying stimulation and it's the most energy-saving option. However, variations of neural tissue impedance make it difficult to control the actual charge that is injected on the tissue, which can lead to safety concerns. This type of stimulators is currently used in techniques such as Deep Brain Stimulation and devices as pacemakers, where proven efficacy and low power consumption are dominant arguments. [10]

Current-Controlled stimulators apply a current stimulus to the neural tissue which is also controlled by the voltage difference between the electrodes. Difference lays in the fact that in this case voltage difference may vary along stimulation time so that wanted current intensity, or current intensity variation, is applied. However, to obtain this type of control, a higher power waste occurs, compromising not only battery lifetime, which is of great importance in portable implantable devices, but also surrounding tissue damage due to heating. This type of stimulators is used, for example, in Transcranial Direct Current Stimulation, a recently used technique whose effects and applications are still under research. [10]

Finally, Charge-Controlled stimulators are a type of stimulators which basically stimulate with a fixed amount of charge via voltage stimulation. For example, a capacitor can be added to a voltage-controlled stimulator for controlling and limit the charge delivered to the electrodes. This is a better energy-efficient type of stimulator than current-controlled stimulators but worse than voltage-controlled ones and it is not yet a clinically-used technique. [10]

Electrical stimulus application can be used through different strategies which can lead to different results due to ion channels conductance and opening rates time dependence, thus making this an area of current research interest. There are some considerations and limitations that need to be considered when using electrical stimulation, such as selectivity, efficacy, efficiency, electrode size and safety. Main safety concerns include biocompatibility, electrode dissolution and creation of toxic chemical substance, already explained in previous section. Since the last two referred safety concerns are related to control of amount of charge passing through the electrodes, one important area of research in electrical stimulation is the design of safe circuits which nowadays make use of blocking capacitors. Although this solution has proven effective, blocking capacitors have an unattractive size and weight for portable and implantable devices that lead investigation towards capacitor-free circuits. Another safety limitation subject of current research areas is related to stimulation selectivity dependency on electrode size and electrode point of application. Electrode design is directly related to selectivity where smaller electrodes lead to higher stimulation precision but, on the other hand, safety charge levels decrease as electrode surface area reduces. Electrode point of application is also an important issue since electrodes close to target neuron provide higher precision but close-located electrodes require a higher biocompatibility level. [10]

Regarding electrical stimulation consideration of selectivity, efficacy and efficiency, besides the dependency on electrode size and point of application, the dependency on stimulation strategy, more specifically waveform shape, amplitude, frequency or width, among others, is also very important and research has shown that more advanced waveforms lead to better results [16]. In matters of waveform shape, the rectangular pulse, presented in Figure 2.8, is the most used waveform shape in neurostimulation because it provides an effective electrical stimulation of simple implementation, however it has several limitations such as the activation of large-diameter fibres before the smaller ones and indiscriminates stimulation of sensory and motor neurons [16].

More advanced waveform shapes, such as the Quasi-Trapezoidal and Tri-Exponential waveform, can lead to more accurate neuron stimulation. Quasi-Trapezoidal is a waveform with a steep rise followed by an exponential decay as shown in Figure 2.9. Tri-Exponential is a triangular-shaped waveform which consists of a linear rise followed by an exponential decrease, as it is presented in Figure 2.10. These waveforms shapes have been found to activate first small-diameter fibres before large fibres and besides

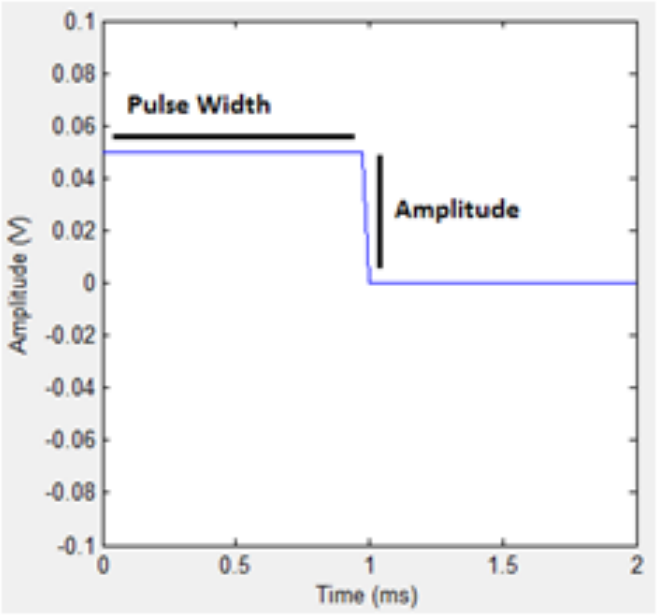

FIGURE 2.8: Rectangular Pulse.

this fibre recruitment characteristic [17, 18], quasi-trapezoidal has also shown success in unidirectional stimulation [19].

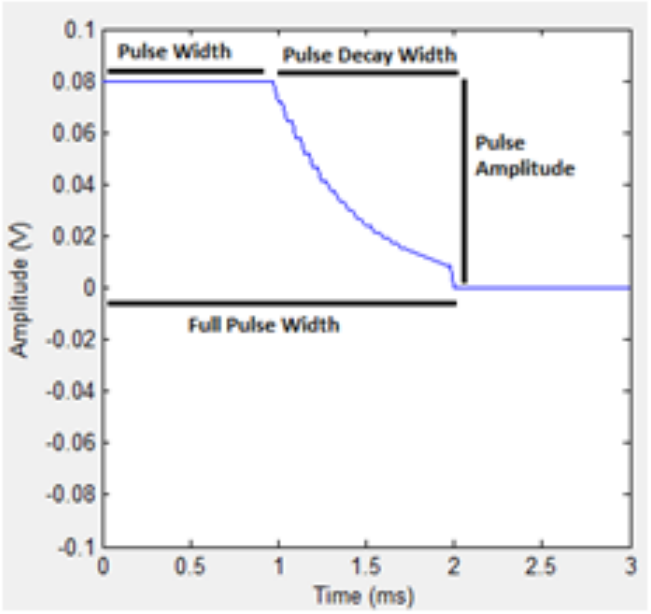

FIGURE 2.9: Quasi-Trapezoidal Pulse.

Activation of medial fibres without activation of lateral fibres is also possible through the use of sub-threshold pre-pulses [20]. An example of a stimulation using this type of strategy is shown in Figure 2.11.

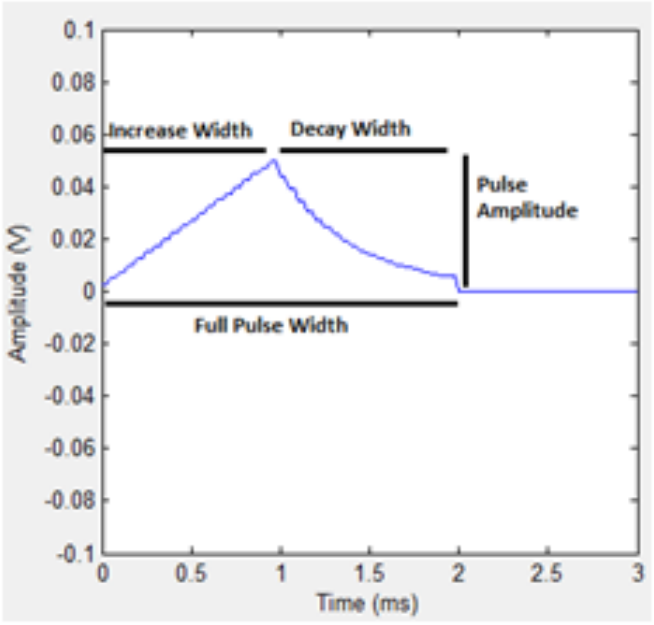

FIGURE 2.10: Tri-Exponential Pulse.

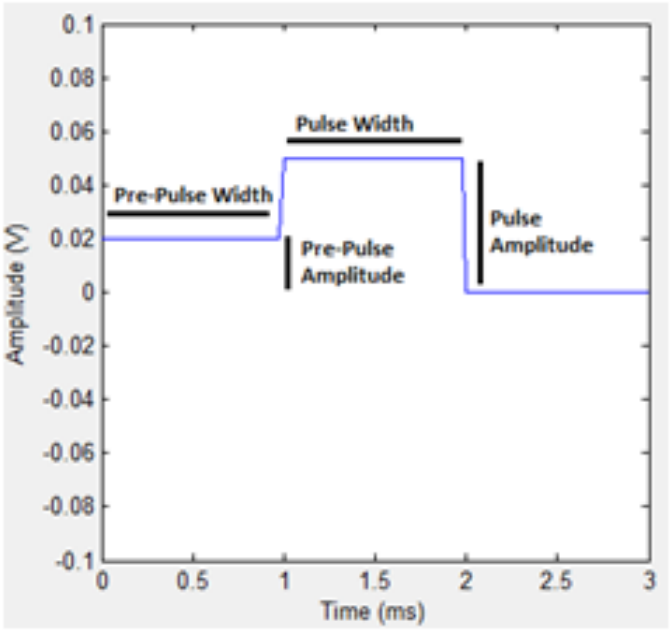

FIGURE 2.11: Sub-threshold Pre-Pulse followed by Pulse.

Other stimulation strategy with good results in selectivity is called bursting strategy, shown in Figure 2.12, which consists of applying smaller rectangular pulses, also called 'pulsons', with a small time interval between them instead of a single rectangular pulse. This bursting strategy is understood to improve not only fiber-selectivity but also power efficiency [21, 22].

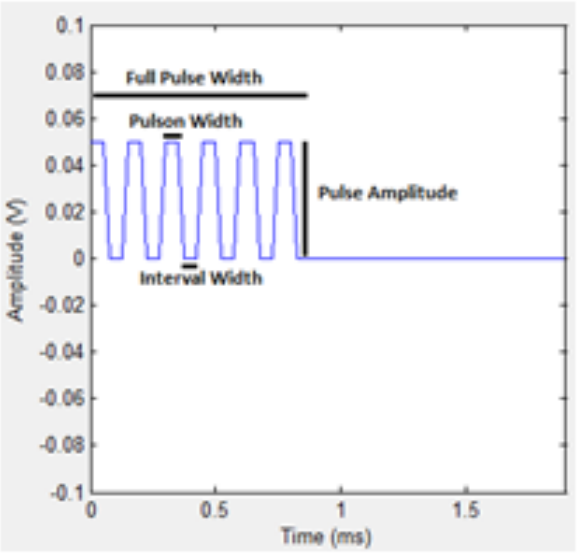

FIGURE 2.12: Bursting Strategy Stimulation.

In summary of waveform parameters, amplitude results in fiber-selectivity, more specifically, the activation of larger fibres with low amplitudes and the blocking of larger fibres previous to smaller ones when an anodal current is applied [17]; frequency causes blocking effects on selected fibre diameters when a high frequency stimulation is applied [23]; and shorter pulse width activates more small-diameter fibres than large ones [24].

Chapter 3

State-of-the-Art of User Configurable Neural Stimulation Technologies

Electrical stimulus application can be used through different strategies for different results due to physiological characteristics of the neuron. When defining a strategy, performance such as selectivity and efficacy, and limitations such as efficiency and safety need to be taken in consideration. The challenge of optimizing the performance of the neurostimulator within the limitations for each specific application is what makes this an area of current research interest.

Towards this goal, research has focused in studying and investigating different types of stimulation strategies in various areas where neural stimulation can be applied. However, with the use of typical neurostimulation experimental methods, changing stimulation strategies is a very time-consuming task and it would not be possible to make a valid comparison between stimulations applied to the same nerve since nerve fatigue could occur and it had to be tested in a new nerve. This way, to allow for further progress in this area, a system that would enable a fast and easy reconfiguration of stimulation parameters was necessary.

In order to respond to this need, new experimental procedures were developed such as the one presented by Grill, W. M., Jr. and J. T. Mortimer (1996) [24] which bypasses this problem through the use of a highly integrated computational model, or the one

presented by Accornero, N., et al. (1977) [18] which uses a common testing platform that integrates a square pulse into a triangular pulse with multiple customized circuits, more specifically, a potentiometer and a switch connected to three different capacitors allowing a control of both amplitude and duration of the triangular pulse. However, in this last procedure it still takes some time to change the configuration, and consequently the parameters, and would also only allow evaluation of a limited number of variables.

Bugbee, M., et al. (2001) [25] developed a programmable implantable stimulator system for selective stimulation of nerves. This system is divided in two parts: one internal and one external. The internal part is placed near the stimulation site and is responsible for the application of stimulation. It is composed by two main integrated circuits (ICs): a Digital Control Unit (DCU) and a Stimulation Unit (SU). SU is the responsible for application of stimulation according to commands from DCU, which receives and decodes radio-frequency information from the external part. The external part of this system is then composed by a computer, a microprocessor-based modulator and a radio-frequency (RF) transmitter. On the computer runs a program that controls the whole system and defines certain parameters for stimulation such as the amplitude, the charge balancing ratio, pulse width, frequency and duration of stimulation. After all parameters are chosen, the program sends that information to the internal part, via the microcontroller and RF transmitter, and then the user can initiate and stop stimulation and save the history of stimulations. This system presented itself has a great advance towards fast and easy reconfiguration of parameters of stimulation and also has the major advantage of allowing programmability with the stimulator already implanted. However there are still limitations namely in the number of parameters that the user is allowed to change and also the options within those parameters. A similar device is reported by Sha, H., et al. (2005) [26] but communication between internal and external part is made via optical transcutaneous communication instead of RF transmission and also the internal part can be composed of more than one stimulator, interacting with only one at a time depending on identification information contained by the light pulse. Here, however, the number of parameters that can be changed is not only limited but also much smaller – about 6 parameters. In Salmons, S., et al. (2001) [27] an analog implantable system is made, but here light is used to select a pre-programmed set of stimulation parameters, which can only be done while the stimulator part is not yet implanted. This therefore means that the number of parameter variations to be tested is limited and to try different

sets or combinations of parameters an extraction and new implantation have to be done, which a very time-consuming, intrusive and nerve-exhausting procedure.

Constandinou, T. G., et al. (2008) [28] presents an integrated circuit for electrical stimulation which allows user programmability of three bipolar stimulation channels. A continuous interleaving sampling strategy is applied which involves sequencing the stimulation waveforms among active channels. This is a current-controlled stimulation system which allows for programmable asymmetric biphasic stimulation where charge-balancing is obtained through a current-steering circuit that has three circuit configuration phases: one for cathodic pulse, one for anodic pulse and a third one, which occurs between pulses, in which current is not directed to the electrodes but instead to a dummy load. This dummy load phase is used to achieve a smoother transition between pulses by minimizing charge buildup and spiking.

Woods, V. et al. (2008) [29] proposes an experimental setup which consists of a software-controlled automated stimulation-and-recording module. This setup allows the user to specify a number of waveform shapes and the corresponding range of stimulation parameters to be investigated, such as amplitude and duration, through a MATLAB-based interface after which the software starts loading the first waveform into a programmable AC current generator. The current output is converted into voltage using an instrumentation amplifier and is then connected to a data acquisition module. To this data acquisition module is also connected the neural recording amplifier. This way, for each stimulation applied, corresponding to the waveform programmed, an acquisition from the nerve is made and stored. After one stimulation and recording is done, the system updates the following programmed waveform and, after allowing some time for nerve to recover, it applies respective stimulation and recording. This system then presents itself with the advantages of re-configurability, allowing an automatic assessment of different stimulation strategies with very rapid parametric variations on one nerve, and portability. However it still presents some limitations. The high price of main components presents as a limitation since to reproduce this system a programmable AC current generator is needed, whose cost is usually above 6.000€, and also a data acquisition module, whose cost is above 2.000€. Also, although portable, generator and acquisition module are still heavy and bulky instruments. Adding to these limitations is the high power consumption they have even when no stimulation is being applied.

Thurgood, B. K., et al. (2009) [30] presents a stimulation system which can be programmed via wireless transmission. Here the programming is made in a computer, where certain parameters can be changed such as current pulse amplitude, duration, interphasic delay and repetition rate, and this information is sent directly from the computer to the stimulator chip. The system was developed with an array of 100 stimulators and allows programming and control of an individual stimulator at a time. The array of stimulators itself is powered via inductive coils and it is effectively controlled by a global finite state machine (FSM), which is the one that receives and interprets the information from the computer to send it to a stimulator. The stimulator itself is composed of analog and digital components that, after receiving and storing information on referred parameters, make it able, on activation, to apply current-based biphasic stimulation. The system developed here presents the major advance of using wireless, an easy, reliable and widely-used means of data transmission, to communicate with stimulators which lead to an easy, remote and hence isolated configurability, meaning it can be a fully implantable programmable stimulator. However, this system has a very limited number of parameters available to configuration, which can be related to a smaller memory available due to size limitations that an implantable device imposes.

Eftekhar, A., et al. (2009) [31] presents a platform that follows the system presented by Woods, V., et al. [29], but instead of using heavy and bulky instruments such as the AC current generator it designs a small PCB-based stimulator which will be the responsible for generation and application of stimulation. This platform is also software-controlled through a MATLAB-based graphical user interface (GUI) that allows specification of stimulation parameters. The interface created in this article allows the user to choose any waveform type desired not only by presenting a number of predefined waveform shapes but also by allowing the upload of arbitrary waveforms. Besides this, the interface allows to specify parameters such as repetition rate, resolution to built stimulation and also amplitude and duration. The computer is then connected to a PCB-based stimulation module composed by a microcontroller, a digital-to-analog converter (DAC), a voltage-to-current converter and an H-bridge, which are supplied by a standard 9 V battery. In this platform, after stimulation profiles are defined in the computer, they are uploaded to the module where they are stored. To choose a stored profile and use it for stimulation a DIP switch is used, following which a trigger input is used to generate and apply the stimulation. This platform then presents the programmability and current-based

stimulation advantages from the previous device but adds portability, since it not only is a relatively small module but also has a battery power supply and, after the profiles are updated, it can work alone without being connected to the computer. Also, building this platform is relatively easy and much cheaper than obtaining the programmable AC current generator. However, this platform presents one major limitation compared to the previously mentioned system: it can only perform stimulation; neural recording is not possible with this module. Besides this, there are also limitations to the number of channels to apply stimulation – it only has one channel which allows up to a tripolar montage – and it also has lower parameter ranges due to microcontroller memory size and other electronic components limitations, such as voltage compliance in stimulation amplitude. Guilvard, A., et al. (2010) [32] make an approach to build a system with capacity of working through multiple channels and allow multi-polar montages, however this system was designed and simulated but fabrication and testing remains undone.

Laotaveerungrueng, N., et al. (2010) [33] present an 8-channel integrated current and voltage stimulator chip. This stimulator chip allows the user to choose a set of stimulation parameters among which stands out the possibility of choosing to apply either a voltage-controlled stimulation or a current-controlled stimulation, which can be programmed independently for the eight channels. The communication between programmer and stimulator is made via SPI where user chosen information besides stimulation control type is sent, such as pulse width, frequency and amplitude. Waveform shape however is not a controllable option similarly to what occurs with interphasic and biphasic ratio. Regarding waveform shape, this system only applies square-shaped waveforms. Regarding interphasic delay, it is defined to be one clock cycle. And with biphasic ratio, besides the fact that the system allows for user to choose between monophasic or biphasic stimulation, it does not allow user to control the degree of symmetry between original pulse and charge-balancing pulse. After all information is sent and stored in the chip, it starts to generate the squared-shaped stimulation according to the referred parameters by sending a digital code identifying the output channel and an analog value of the waveform amplitude.

Chiu, H.-C., et al. (2014) [34] have developed a two-channel micro-stimulating and neural recording brain-machine interface that provides a monophasic voltage-controlled stimulation based on parameters defined by the user through a GUI in a computer, which are transferred to the biomedical core, processed and sent through a DAC to

the stimulator front-end, and also provides detection of neural activity through probes after which it is amplified, converted to digital by a successive approximation register (SAR) analog-to-digital converter (ADC) and passed to the biomedical core where it will be processed and transmitted. In this system user can control specifications from both recording and stimulation. Regarding recording, specifications include operating voltage, channel, sampling rate, bandwidth and gain. Regarding stimulation, controllable specifications are operating voltage, channel, amplitude, frequency, pulse width, and time. This system presents itself has a very useful tool for neural activity and stimulation studies especially for the fact that it allows for a joint operation of stimulation and recording, where it also allows for configurability of both these modalities, and also because it is done via Bluetooth which makes this a better option for studies where implantation is intended. However, in the stimulation side, there are still a number of limitations in particular with the parameters available for programmability and characteristics of stimulation. Regarding parameters, the fact that it is not possible to control the waveform shape is a major limitation as well as the impossibility of using either a monophasic or biphasic stimulation. Regarding stimulation itself, the monophasic-only limitation referred previously also affects here since it is seen as not so safe type of stimulation. Besides this, the fact that it is a voltage-controlled stimulation can also be seen as a limiting factor since it is more difficult to control the amount of charge injected in the tissue, although a more power efficient method which is an advantage since this is an implantable system.

Besides these devices, some were also developed and made available in open-source platforms such as the open-source platform for electrophysiology called Open Ephys [35]. An available device in this platform is the Pulse Pal [36]. Pulse Pal is an open-source device that allows for users to program and generate sequences of voltage stimulation pulses. It presents four output channels, which can be independently programmed and have unique pulse sequences, and two trigger channels that can control the output of the pulses. This device allows user to control a set of 16 parameters, as the voltage amplitude, the width of the pulse, if it is monophasic or biphasic, the interphasic delay, the charge-balance amplitude and width, the interpulse, or resting, voltage and width. It also allows user to use a bursting strategy instead of plain rectangular pulse and program its pulsons width and inter-pulsons width. Besides this, the device also allows configuring the delay between trigger and beginning of output of the pulses and duration of all pulse train and

it is also possible to program the trigger mode. All these parameters can be configured in the device itself, due to the existence of an OLED screen and a joystick, but can also be connected to computer and be programmed from there through a programming interface such as MATLAB, or via C++ language or Python. In case of being programmed in a computer, it is also possible to generate different and more complex pulse trains and then load them to the device. However, due to memory limitations, this option is only available for a maximum of two channels. With all these referred characteristics, Pulse Pal is a very promising tool for stimulation study. Adding to the fact of having a wide range of parameters available for configuration, the availability of trigger mode and the possibility of programming either in stand-alone mode or through a programming interface, it is the fact of being a low-cost device – it costs about $ 200 – and the fact that the software and hardware assembly instructions are openly available. However, there are limitations such as the ones resulting from the memory, which results in only two possible custom-made waveforms, and the fact that only a channel at a time can output.

The electrical stimulation devices that were described in this chapter present characteristics that make them very useful tools for the study of electrical stimulation strategies. However, limitations can still be found in all these systems. The development of a programmable electrical stimulation platform which presents the advantages and overcomes the referred limitations of the reported devices is then a major goal intended for the project here developed.

Chapter 4

Software Development

The development of the stimulation platform is described throughout this thesis and it is divided in the two main parts that compose the system: software and hardware. The software part is responsible for receiving the user's configuration, processing them and then control the hardware part so that stimulation can be done accordingly.

This chapter is dedicated to the software part of the system, where the requirements, the software developed and its operation are described. For this part, two different types of software platforms were used namely MATLAB and CodeWarrior which were used, respectively, to create the user-system interface and to program a FRDM-KL26Z Freescale microcontroller. The requirements, development and operation of each of this software will be done separately, starting by a section for user-system and then a section for microcontroller.

4.1 MATLAB-based User-System Interface

As previously stated, MATLAB was the software platform used for the development of the user-system interface. The choice of MATLAB to do such an interface was not only because it is a very powerful and interactive software for numeric computation, data analysis and visualization but also a very intuitive and easy-to-use platform for programming and algorithm development and also has as specific tool for the design and development of user interfaces called GUI Development Environment (GUIDE). Figure 4.1 presents the MATLAB GUIDE interface.

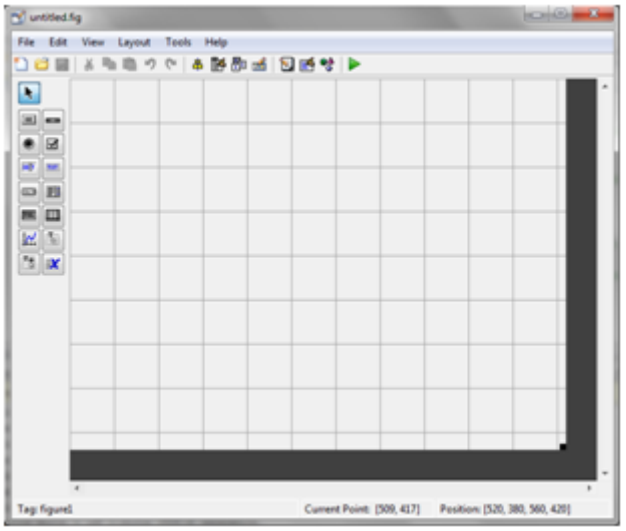

FIGURE 4.1: MATLAB GUIDE interface.

The GUIDE interface allows user to design a GUI with several of predefined components such as pushbuttons, checkboxes, pop-up menus, among others. After placing the wanted components on the GUI design area (grey grid area), MATLAB automatically generates a function to match each of the components placed in a main function that corresponds to running the GUI. Inside these functions can then be written code lines that will define what happens when the respective component is used. Figure 4.2 presents an example of a function generated for a pushbutton.

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

FIGURE 4.2: Pushbutton function code.

To develop the user-system interface, there were a number of requirements that were initially defined as necessary for the program to have. These requirements were essentially related with parameters from stimulation that the user should be able to configure and control. The requirements were the following:

- To be able to choose between different waveforms.
- To allow configuration of pulse parameters, such as amplitude and width.
- To allow configuration of interphasic delay.

- To allow configuration of biphasic ratio.
- To allow configuration of pulse repetition.
- To allow control of stimulation duration.

These were defined as priority requirements since they were already implemented by some of the existing configurable stimulation systems [25-34, 36] as it was referred in Chapter 3.

Besides these priority requirements, other requirements for stimulation configuration were also taken into account to be available for user control. The first requirement is related to the control of existing stimulation channels. More specifically, it was desired that this interface would allow user to control four stimulation channels, that each one could be programmed independently and also that it would be possible to use these channels either stimulating individually, stimulating at the same time – synchronously – or stimulating sequentially with a fixed delay between them – asynchronously.

The second requirement was related to voltage amplitude limitations of this stimulation system, more specifically, to the limitations of front-end stimulation circuit. The purpose of this requirement is to warn the user of the maximum amplitude that the stimulation system is able to sustain in order to apply stimulation as user configured in the conditions that stimulation is going to be applied, in this case in particular the electrode impedance.

The following sections provide a description of the user-system interface functionalities created to accomplish the requirements aforementioned.

4.1.1 Choosing between Different Waveforms

Regarding to the first point of requirements, the ability to choose between different types of waveform, it was defined that the available waveforms would be the ones used by Eftekhar, A., et al. [31], which are: Bi-level Lilly, Quasi-Trapezoidal, Tri-Exponential and Bursting Strategies. Besides these four options, it was also included a simple Rectangular pulse thus having a set of five pre-selected waveforms. Besides this five waveforms, it was also included a sixth option which allows user to upload an arbitrary waveform of its choice. To allow the user to perform this selection, a pop-up menu with all the

referred options was created, as can be seen in Figure 4.3.

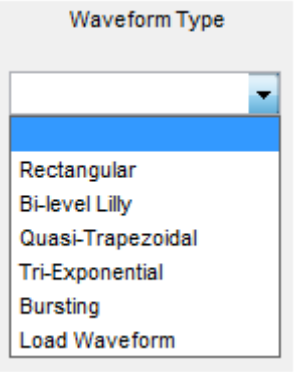

FIGURE 4.3: Waveform Type pop-up menu.

For this pop-up menu, the function written was that when the user chooses one of the five preset waveforms, a one pulse stimulation using the matching waveform is built and saved in an array. Here, the basic necessary parameters are pre-defined to build the respective waveform: amplitude, width, biphasic ratio and interphasic delay. This is done so that user can have an idea of the waveform chosen without having to configure all these referred characteristics. The building of one stimulation pulse is made by concatenation of three arrays – Equation 4.1.

$$pulse = charge\,phase + interphase\,delay + charge\,balancing\,phase \tag{4.1}$$

The first array, which is for charge phase, results of multiplying an array of ones, with length equal to width T_O, in milliseconds (ms), times 40, by the amplitude value A_O, in milliVolts (mV) – see Equation 4.2.

$$charge\,phase = A_O * ones[1, T_O * 40] \tag{4.2}$$

The second array, which is from interphasic delay, is a zero array whose length corresponds to the duration of interphasic delay T_{ID} – see Equation 4.3.

$$interphasic\,delay = zero[1, T_{ID} * 40] \tag{4.3}$$

The final array, which is from charge-balancing phase, is obtained similarly to the first one, with the key differences being the fact that here the charge-balancing amplitude A_{CB} has opposite sign than the original, and the fact that charge-balancing amplitude and width T_{CB} used are obtained by the division and multiplication, respectively, of the original value by the biphasic ratio B – Equation 4.4. Width is multiplied by 40 because it was defined that one array element corresponds to 0.025 ms, hence 40 elements correspond to 1 ms.

$$charge\,balancing\,phase = A_{CB} * ones[1, T_{CB} * 40] = \frac{-A_O}{B} * ones[1, B * T_O * 40] \quad (4.4)$$

Here, *ones* and *zeros* are MATLAB functions that generate arrays filled only with one and zero, respectively, whose number of lines is given by the first value inside square brackets and columns by the second value. In this interface it is pre-defined that charge phase, also called cathodic, is represented as negative amplitude and that charge-balancing, also called anodic, is represented by positive amplitude. The equations above presented are the basis of almost every edit box created for parameters, however there are changes in the way charge phase array is calculated depending on the waveform chosen, as it will be explained further. Figure 4.4 shows a rectangular waveform with all the above mentioned phases.

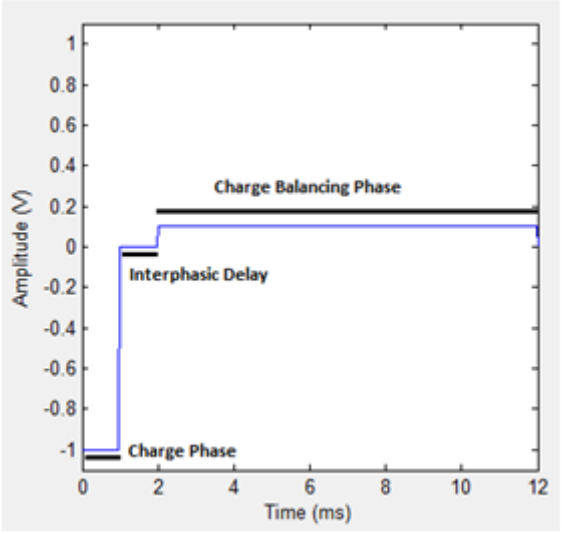

FIGURE 4.4: Rectangular waveform-based pulse.

4.1.2 Configuring Pulse Parameters

Concerning configuration of pulse parameters, edit boxes were created to allow the user to input the desired values. Input boxes were created for control of both pulse amplitude and width. However due to the characteristics of some of the preset waveforms, other input boxes had to be created besides these two. Firstly, we had to create two more edit boxes for amplitude and width for Bi-level Lilly waveform since this waveform can have two different amplitude levels. To differentiate the two, the amplitude edit box that applies to all waveforms is identified as "Primary Pulse Amplitude" and the width is identified as "Primary Pulse Width", and the amplitude and width edit boxes that only apply to Bi-level Lilly waveform are respectively identified as "Secondary Pulse Amplitude" and "Secondary Pulse Width". The function activated when primary amplitude and width edit boxes have a new input is similar to the one from waveform type pop-up menu but here the variable of charge phase amplitude and width, A_O and T_O, have the input value. When secondary amplitude and width edit boxes have an input a similar process to the one for primary parameters occurs. The difference between primary and secondary calculation is the addition of one phase array, which follows the existing charge phase. This phase array is also a result of multiplying an array of ones – with length equal to secondary width T_{O2}, in milliseconds (ms), times 40 – by the secondary amplitude value A_{O2}, in milliVolts (mV). Besides this, there is also inclusion of both these secondary parameters in the calculation of amplitude and width of charge balancing phase. To avoid any possible error with the remaining waveforms, these edit boxes are locked and respective variables are set null whenever Bi-level Lilly is not selected. Figure 4.5 shows the mentioned edit boxes with the pre-defined values. Figure 4.6 shows the same edit boxes but with secondary parameters locked.

FIGURE 4.5: Amplitude and Width edit boxes.

Figure 4.7 shows a pulse built with use of Bi-level Lilly waveform type with the preset parameters shown in Figure 4.6.

FIGURE 4.6: Amplitude and Width edit boxes with secondary parameters locked.

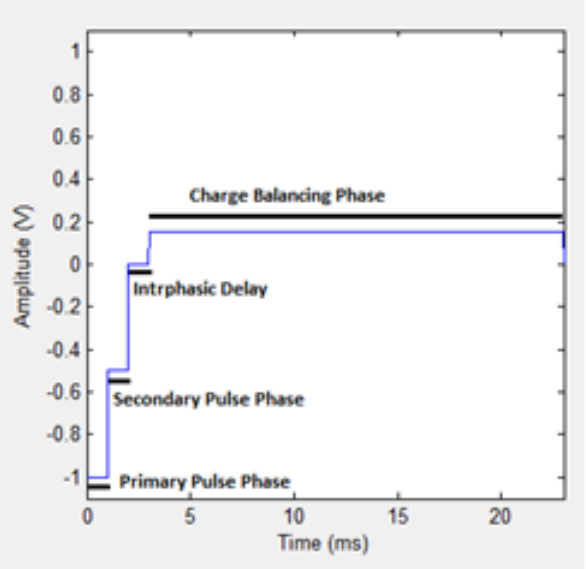

FIGURE 4.7: Bi-level Lilly waveform-based pulse.

Secondly, considering the characteristics of Quasi-Trapezoidal and Tri-Exponential waveforms, more specifically the fact that both these waveforms have an exponential decay, edit boxes for configuration of exponential components were also created, thus giving the user control over the shape of the decay. However, there is a difference between the decays of the referred waveform, this difference being the fact that Quasi-Trapezoidal decay is exponential while Tri-Exponential decay is, has the name implies, tri-exponential. Figure 4.8 and 4.9 show the edit boxes created respectively for Quasi-Trapezoidal and Tri-Exponential waveforms.

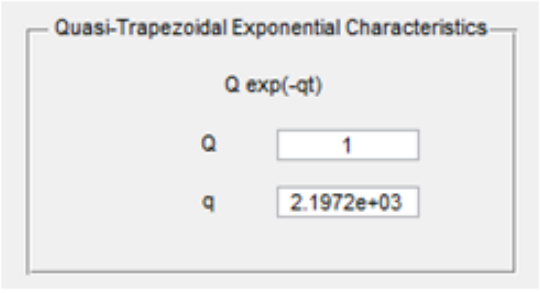

FIGURE 4.8: Edit boxes for Quasi-Trapezoidal exponential characteristics.

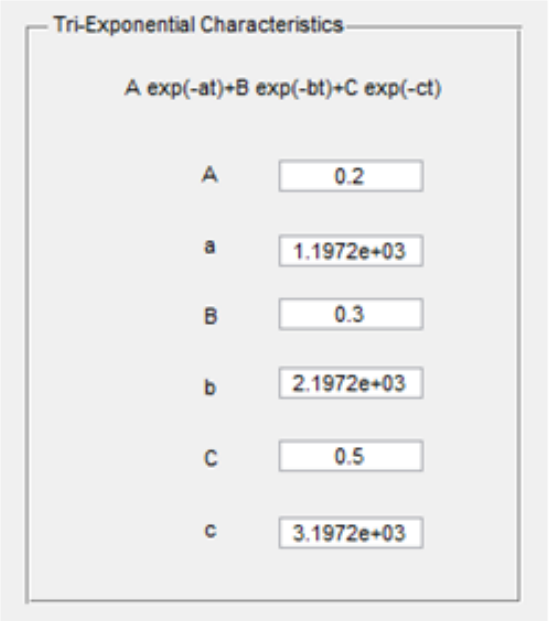

FIGURE 4.9: Edit boxes for Tri-Exponential exponential characteristics.

As it is possible to see in the figures above, an edit box was created for each parameter of the exponential, whose identification is explained in the equation presented above the respective edit boxes. Here, as in previous edit boxes, the functions activated when an input is done have the same Equations 4.1, 4.2, 4.3 and 4.4 as its basis. However, in these cases, building of charge phase array is divided in two parts. In Quasi-Trapezoidal case, the first part of this phase is built the same way, as presented in Equation 4.2, using "Primary Pulse Width" as this phase width; the second part of the phase is built through concatenation of smaller arrays, of length equal to time resolution multiplied by 1ms of array elements, composed by amplitude values obtained by iteration of exponential equation starting from the amplitude value, chosen by the user or pre-defined, until a certain limit of number of iteration steps is reached. The equation used for iteration is presented on the top of Figure 4.8 and can be seen in Equation 4.5. The number of iteration steps is obtained from the division of decay width by resolution and so, to allow a higher user-control over the behavior of the decay, and more specifically of number of steps, edit boxes were also created for configurability of decay width and resolution intended – see Figure 4.10.

$$A_d(T_O + t) = Qe^{-q(T_O+t)} \tag{4.5}$$

In Equation 4.5, the t is the resolution, or time step, and A_d is amplitude in decay phase.

During this second part, summation of amplitude obtained in each iteration is done so that, by adding its result to first phase amplitude, it is possible to calculate correctly the amplitude of charge-balancing phase.

FIGURE 4.10: Decay Width and Resolution edit boxes.

Figure 4.11 shows a Quasi-Trapezoidal pulse with the referred phases. In Decay Phase it is possible to see the iteration steps, which can be smoothen if resolution is configured to be smaller.

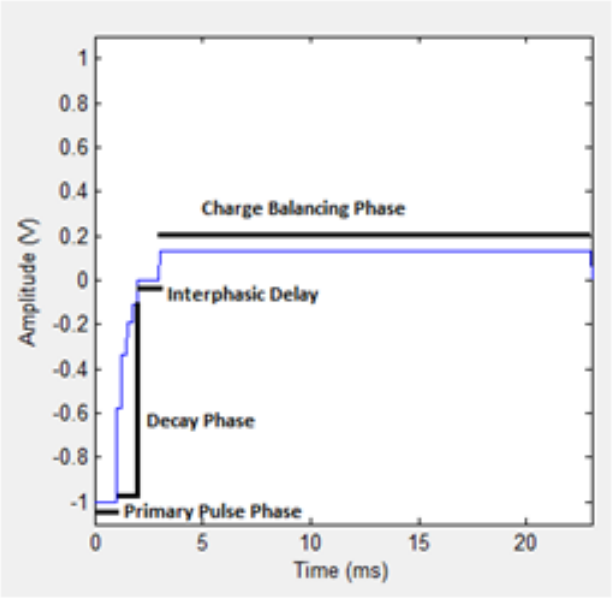

FIGURE 4.11: Quasi-Trapezoidal waveform-based pulse.

Regarding Tri-Exponential case, the first part of charge phase is a linear increase of amplitude so to build this part array Equation 4.6 is used, obtained knowing that the maximum value is the amplitude defined by the user, or pre-defined, that initial amplitude is zero and also that charge pulse starts in beginning, which means when time is zero. Using this equation, iteration is made and all the resolution-length arrays composed of amplitude values, obtained from each iteration, are concatenated together building an array of increasingly higher amplitude values up until it reaches a limit number of iterations. This number of iterations is defined similarly to decay, the difference being

that instead of using decay width value, primary pulse width value is used.

$$A_i(t) = \frac{A_O}{T_O}t + \frac{-A_O}{T_O} \tag{4.6}$$

In Equation 4.6, A_i is amplitude in increase phase and t still is resolution, or time step. The second part of the charge phase, which corresponds to the decay, is built similarly to Quasi-Trapezoidal waveform, meaning it is also obtained through concatenation of smaller arrays, of length equal to time resolution multiplied by 1 ms of array elements, composed by amplitude values obtained by iteration of exponential equation starting from the amplitude value until a certain limit of number of iteration steps is reached. The only difference here, comparing to Quasi-Trapezoidal, is the fact that a tri-exponential equation is used to calculate the decay. This equation can be seen in Figure 4.9 on the top and is presented in Equation 4.7.

$$A_d(T_O + t) = Ae^{-a(T_O+t)} + Be^{-b(T_O+t)} + Ce^{-c(T_O+t)} \tag{4.7}$$

With this waveform, summation of amplitude obtained in each iteration is done, similarly to Quasi-Trapezoidal, but here is done in both part one and two so that by adding both amplitude results, it is possible to calculate correctly the amplitude of charge-balancing phase. Figure 4.12 shows a Tri-Exponential pulse with the referred phases and building characteristics identified.

Besides these two waveforms, it is also important to refer the parameters that can be configured for Bursting Strategies. In here, besides primary pulse amplitude and width, also resolution can be configured since it is an important factor on the building of pulse. In this waveform type, the building of pulse is also done by concatenation of arrays through iteration but in this case each iteration adds an array that can be divided in two parts: the first part is a set of primary amplitude values with length equal to half of the resolution multiplied by 1ms of array elements and the second part is a set of zeros with the same length of the first part. Iteration is done up to a number of steps which is obtained the same ways as in previous waveform, i.e., result of dividing primary pulse width by resolution. Figure 4.13 shows a pulse built making use of Bursting Strategies waveform type.

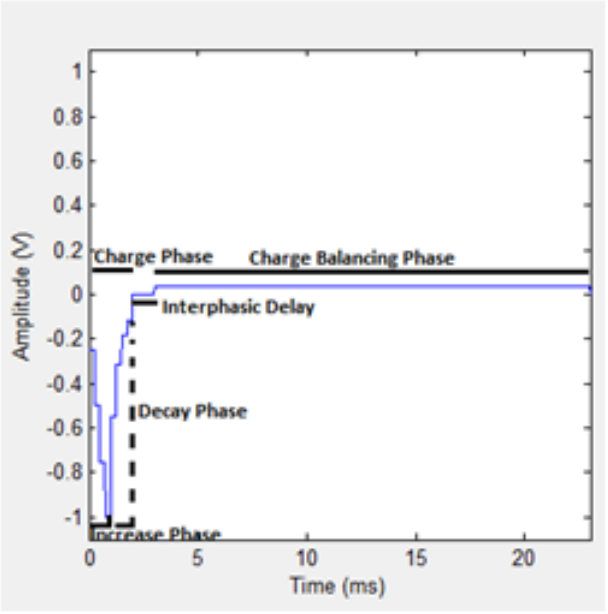

FIGURE 4.12: Tri-Exponential waveform-based pulse.

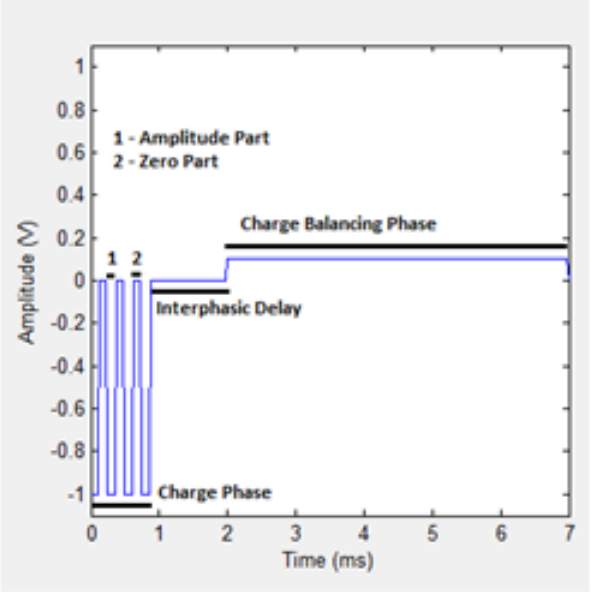

FIGURE 4.13: Tri-Exponential waveform-based pulse.

A common event in the function of all the edit boxes referred is that, when an edit box is used and an input is placed, the initial stored pulse is substituted by a new pulse, built from the beginning, using the new input value and the already stored information, either pre-defined or input, of the other parameters. The pre-defined values for the aforementioned edit boxes can be seen in the figures from edit boxes presented. Similarly to secondary amplitude and width edit boxes, the edit boxes that are specific to a waveform are locked when that waveform is not selected.

4.1.3 Configuring Interphasic Delay

Regarding configuration of interphasic delay, similarly to what was done with the previous parameters, an edit box was created, as it can be seen in Figure 4.14. Equally similar to the boxes, the function behind it is the same as the one behind waveform type pop-up menu, meaning that when a value is input in this box, the pulse is again rebuilt using the Equations 4.1, 4.2, 4.3 and 4.4 according to the new interphasic delay value, taking into account the waveform type selected, which is especially important for the building of charge phase array, and the already input, or pre-defined, relevant parameters.

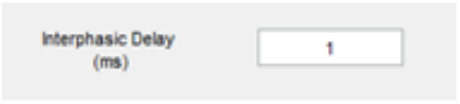

FIGURE 4.14: Interphasic Delay edit box.

The pre-defined value for interphasic delay is, as it can be seen in Figure 4.14, 1 ms.

4.1.4 Configuring Biphasic Ratio

Concerning configuration of biphasic ratio, an edit box was also created as shown in Figure 4.15. The functioning of this edit box is equal to the one from interphasic delay, i.e., the pulse is again rebuilt using the same equations according to the new biphasic ratio value and taking into account the waveform type selected and the remaining relevant parameters already stored.

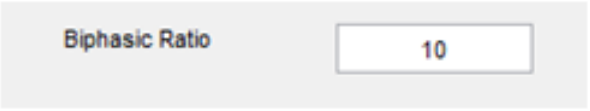

FIGURE 4.15: Biphasic Ratio edit box.

Biphasic Ratio is preset to 10. It is also possible to override Biphasic Ratio in order to have a monophasic pulse by setting it to zero value.

4.1.5 Configuring Pulse Repetition and Stimulation Duration

Regarding configuration of pulse repetition, two configurable parameters were made available: repetition rate and repetition amplitude ratio. Repetition rate is the frequency, in kiloHertz (kHz), at which the whole pulse is repeated throughout the full

stimulation duration. Repetition amplitude ratio is a ratio which is applied to the pulse between every repetition. This allows the user to give a repetitive stimulation with increasingly higher or lower stimulation pulses. For both of these parameters edit boxes were created that allow the user to input desired values and they both work in a similar way, the only difference being the parameter that is changed. Figure 4.16 shows both edit boxes with respective pre-defined values.

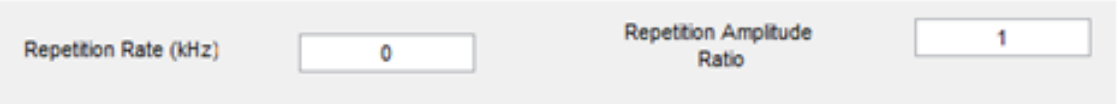

FIGURE 4.16: Repetition Rate and Repetition Amplitude Ratio edit boxes.

The function behind both of these edit boxes is the same and consists of an iteration within which a concatenation of three arrays is done. The first array is the stimulation array which is initially composed by one pulse and is where all the stimulation will be stored; the second array is an array of zeros with length equivalent to the interpulse delay; and the third array is an array of one stimulation pulse multiplied by the repetition amplitude ratio. Interpulse delay is the inverse of repetition rate meaning it is the time between repetitions. The number of iterations done is the result of dividing the duration defined for the whole stimulation by the sum of duration of a pulse with interpulse delay.

Stimulation duration is a very important parameter and can also be programmed through the edit box presented in Figure 4.17. The operation of this edit box is similar to the repetition rate and repetition amplitude ratio, i.e., applying concatenation of the three arrays abovementioned by iteration. The value that is presented in the edit box is automatically calculated whenever a configuration change of the pulse is made.

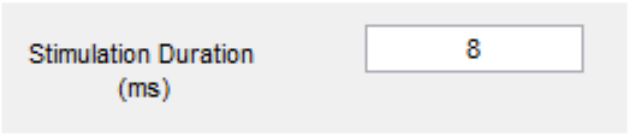

FIGURE 4.17: Stimulation Duration edit box.

The real effect of these parameters on the stimulation profile being configured is however not done in the computer but instead in the microcontroller. In fact, the array here built serves only to create the stimulation profile so that user can see the result. The most important outcome of using these parameters is the number of repetitions, the interpulse

delay and repetition amplitude ratio, information which will be sent to microcontroller and whose use will be explained further, in Section 4.2.

To allow the user to have a better insight on the pulse that is being programmed, a set of axes was also included in the user interface, which responds to when waveform type is changed and is updated when user presses a pushbutton identified as "Plot". Figure 4.18 shows both the axes area and the plot pushbutton created.

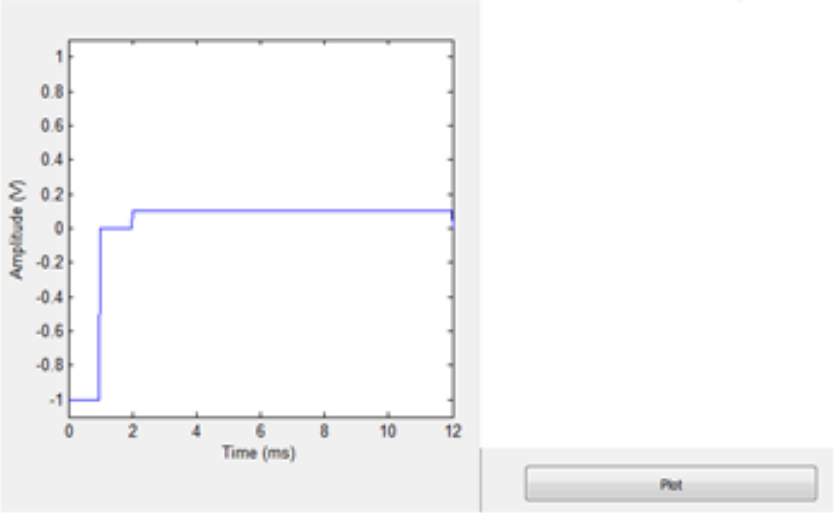

FIGURE 4.18: Axes area and Plot pushbutton.

4.1.6 Configuring Channels

As was mentioned previously, the system here presented is composed by four channels therefore it is of utmost importance that the user-system interface has the capacity to allow the user to control all available stimulation channels. To allow an easy configuration of any of the channels independently, a pop-up menu was created that would allow the user to choose the channel through which the configured stimulation will be applied. Figure 4.19 shows the mentioned pop-up menu.

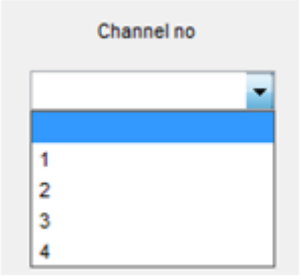

FIGURE 4.19: Channel pop-up menu.

The change of channel does not lead to any changes on the rest of parameters present but instead informs the microcontroller which channel is being programmed.

Another important parameter is control over the way stimulation channels are used, or in other words, the firing type. To this end another pop-up menu was created, as shown in Figure 4.20. This menu allows the user to choose between three different uses of stimulation channels: alone stimulation, synchronous stimulation and asynchronous stimulation.

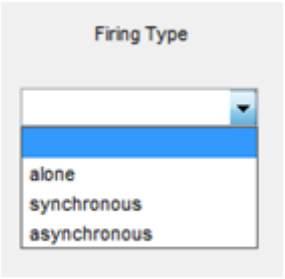

FIGURE 4.20: Firing Type pop-up menu.

Alone stimulation consists of using a single channel to apply stimulation, which can be any of the four channels available. Synchronous stimulation is a stimulation firing type where up to four channels can be used and consists of starting to apply the stimulation at the same time in all chennels. Asynchronous stimulation is a type of stimulation firing where users can use up to four channels and its operation consists of applying the stimulation one channel at a time, sequentially, from Channel 1 to Channel 4, with a delay, or time lag, between them. To allow even more configurability of this firing type, an edit box to configure this delay, in milliseconds (ms), was also created – see Figure 4.21.

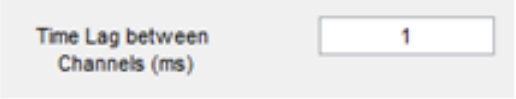

FIGURE 4.21: Time Lag edit box.

It is also important to mention that, in both synchronous and asynchronous firing types, stimulation can be based on different waveforms, different amplitude values and a different number of repetitions. In the synchronous case, however, the repetition rate of stimulations in all the channels used has to be the same.

These firing type definitions don't have any effect on the rest of the waveform parameters, except for the fact that whenever asynchronous is not selected the Time Lag edit box is locked, and its information is sent to the microcontroller.

4.1.7 Configuring Maximum Amplitude Voltage

Due to some electronic limitations of the front-end circuit, which will be described later in Chapter 5, this stimulation system has a maximum amplitude voltage that can be applied during stimulation, beyond which the stimulation profile will not be output as programmed. This maximum voltage is determined to be 5 V but it can vary, and its variation is mainly dependent on the output impedance. Here, the biggest influence comes from the electrode impedance which varies from electrode to electrode, depending on characteristics such as electrode material and size. Hence, with this in mind, a panel with two edit boxes was created: one for user to input the impedance of electrode that is going to be used, if known; a second one, which is locked and not available to be edited, that will only be used to present to the user the calculated maximum amplitude voltage. This panel can be seen in Figure 4.22.

FIGURE 4.22: Electrode Impedance panel.

Due to the importance of this limitation, an additional measure was taken, which consisted in programming a dialog box that warns the user for this limitation and allows him to input the electrode impedance value – see Figure 4.23. In case the user inputs the impedance, the respective values will appear on the panel shown in Figure 4.22. Otherwise, the warning window in Figure 4.24 will appear, to warn the user that maximum amplitude was not calculated.

The calculation of Amplitude Maximum from the Electrode Impedance value is done through Equation 4.8.

$$A_{max} = \frac{A_{max,def} * R_{set}}{Z + R_{set} + 2R_{internal}} \tag{4.8}$$

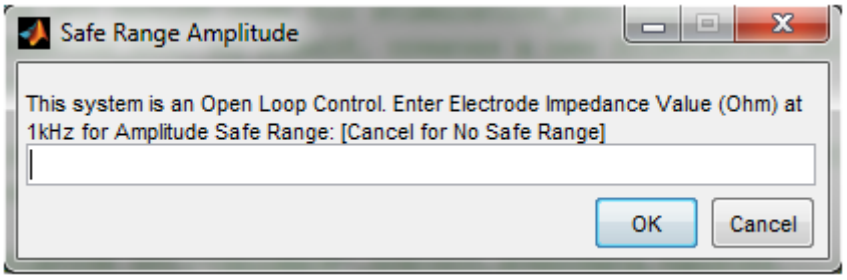

FIGURE 4.23: Electrode Impedance dialog window.

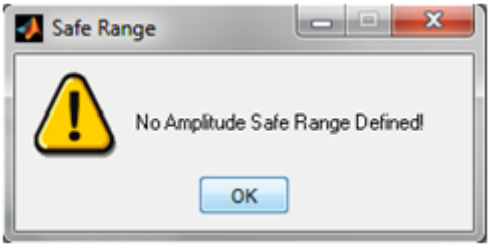

FIGURE 4.24: Warning window when no Electrode Impedance value is input in dialog window.

In Equation 4.8, A_{max} represents Maximum Amplitude calculated, $A_{max,def}$ is the maximum amplitude by definition, both in Volt (V), Z is the electrode impedance, R_{set} is the value of set resistor, which will be explained further in Chapter 5, and $R_{internal}$ is the resistance from the switches used, the three of them in Ohm.

4.1.8 Overall Operation and Communication with Microcontroller

The resulting interface is shown in Figure 4.25 and, as it can be seen, presents all the requirements and respective components referred in the last sections.

As can be seen in Figure 4.25, there are three pushbuttons that were not referred previously. This was because these pushbuttons are not related to the configurability of stimulation profile but instead are used for communication with the microcontroller. The diagram shown in Figure 4.26 represents the operation of the MATLAB-based user-system interface.

The diagram from Figure 4.26 shows that there are three configurations that are of most importance to be chosen first, which are the Firing Type, Waveform Type and Channel Number. While the information on the first and the third configurations are simply stored to be sent later to the microcontroller, the Waveform Type configuration is, as said before, associated to other parameters available for configuration. As can

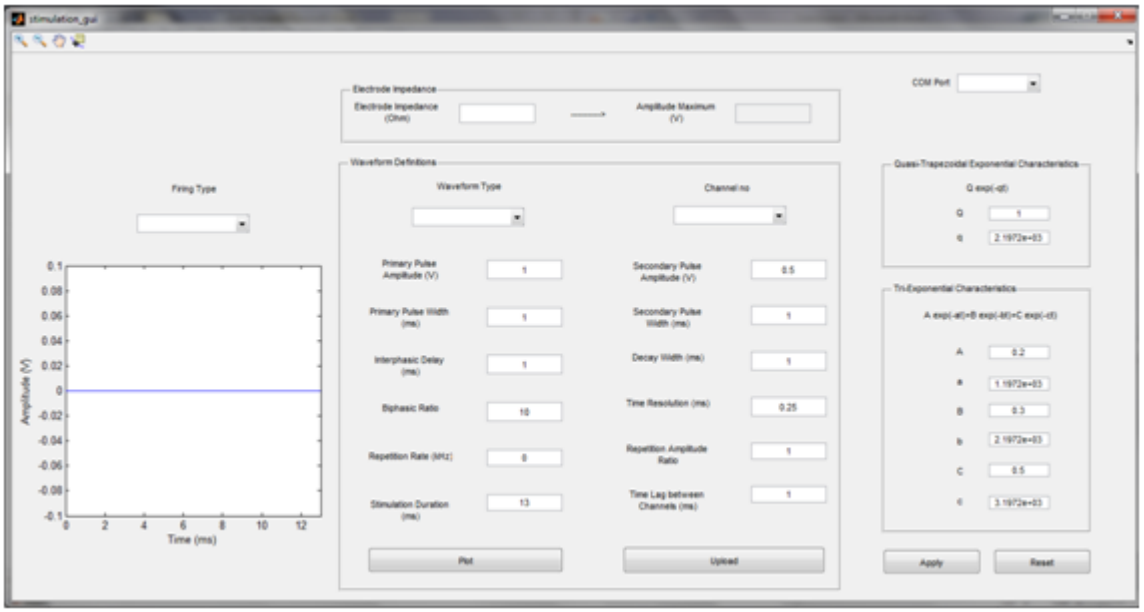

FIGURE 4.25: Full User-System Interface.

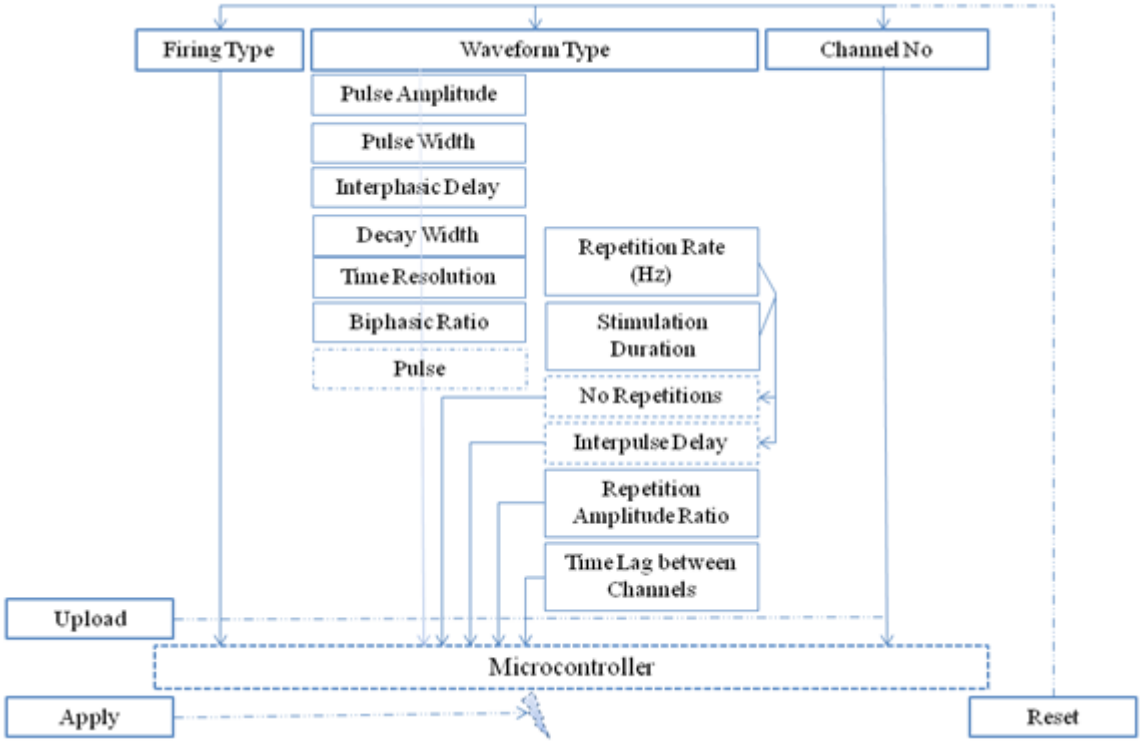

FIGURE 4.26: User-System Interface Operation Diagram.

be seen in the diagram, after choosing the waveform type, a single stimulation pulse is built based on values defined for parameters pulse amplitude, width, interphasic delay, decay width, resolution and biphasic ratio, either pre-defined or defined by the user, which is then stored. Parameters such as Repetition Rate and Stimulation Duration are used to obtain the number of repetitions and the interpulse delay which are then

stored. Also stored with no processing are the values of Repetition Amplitude Ratio and Time Lag between Channels. This stored information can then be transferred to the microcontroller by pushing "Upload" button – see Figure 4.27. When this pushbutton is pressed, MATLAB starts a communication protocol which consists of signaling the piece of information that is going to be sent and then sends it. It is important to refer that, regarding pulse upload, MATLAB sends the absolute values of the pulse and, instead of sending negative values, it sends a signal to the microcontroller of when in the pulse there is a variation from negative to positive, or vice-versa, or a variation to or from zero. The state of the transfer of information is transmitted to the user by a waiting bar, present in Figure 4.28.

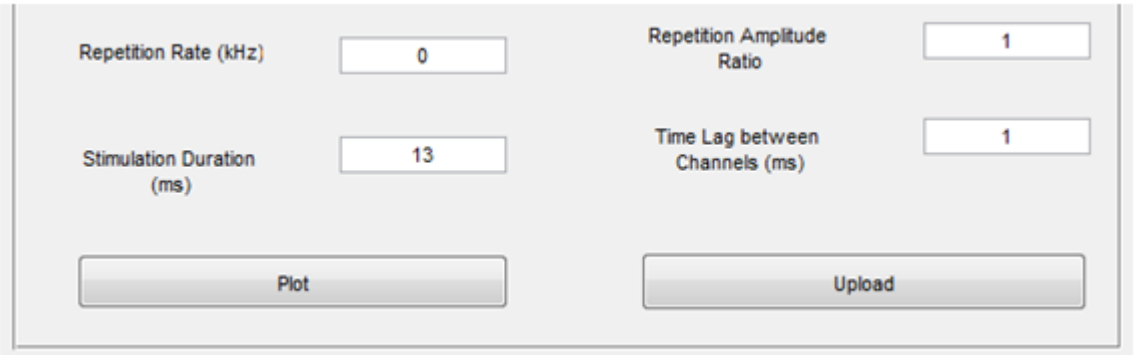

FIGURE 4.27: Upload pushbutton.

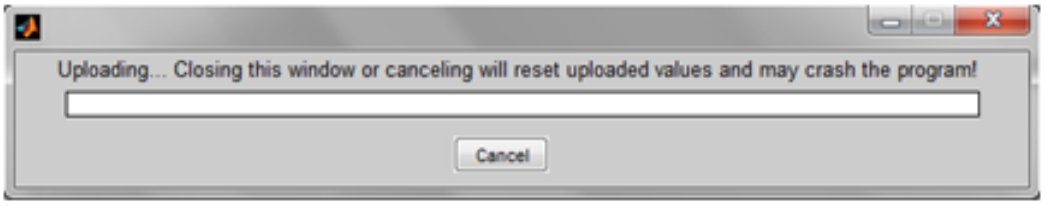

FIGURE 4.28: Waiting Bar window.

After sending all the information, the user can then decide to upload pulses for the other channels, if the firing type is either synchronous or asynchronous, or change the pulse programmed for the same channel. Furthermore, the user can decide if the configurations done are enough and choose to apply the respective stimulation or decide to clear all the information stored in microcontroller. If user decides to apply, the "Apply" pushbutton has to be pushed after which stimulation is applied and the information existent in microcontroller is subsequently cleared. If user decides to clear all the information in the microcontroller, the "Reset" pushbutton has to be pushed, after which no information will remain stored in microcontroller. Figure 4.29 shows these last referred pushbuttons.

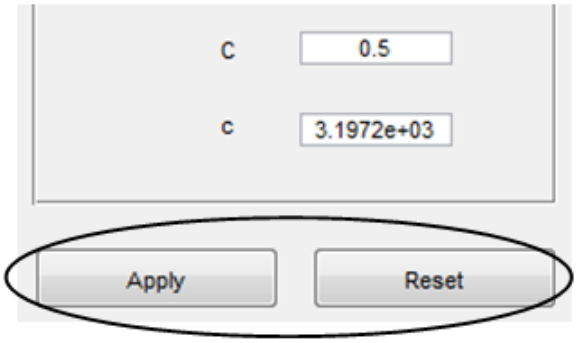

FIGURE 4.29: Apply and Reset pushbuttons.

To allow for a better operation of this user-system interface, other pop-up menu was also included identified as "COM Port" – see Figure 4.30. The reason behind the creation of this menu was the fact that connection of microcontroller to COM ports from computer was many times in the origin of problems related with sending information to the microcontroller, more specifically its misidentification. The pop-up menu here created only shows the COM ports being used at the time and this way allows user to easily change the port or correct if the wrong one is being used.

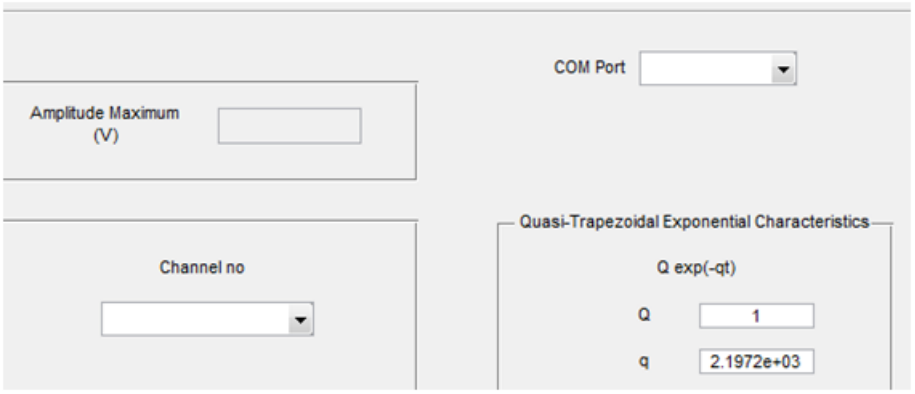

FIGURE 4.30: COM Port pop-up menu.

4.2 *Freescale* Microcontroller FRDM-KL26Z

As already mentioned in the previous chapter, the role of microcontroller is to receive the information about stimulation parameters from MATLAB and then use it to control a front-end stimulation circuit in order to produce stimulation as desired by user. The microcontroller used for this work was a *Freescale* FRDM-KL26Z which was programmed through an integrated development environment (IDE) called CodeWarrior that allows

for software writing, compiling and debugging in C programming language. A diagram representing the operation developed for the MCU is presented in Figure 4.31.

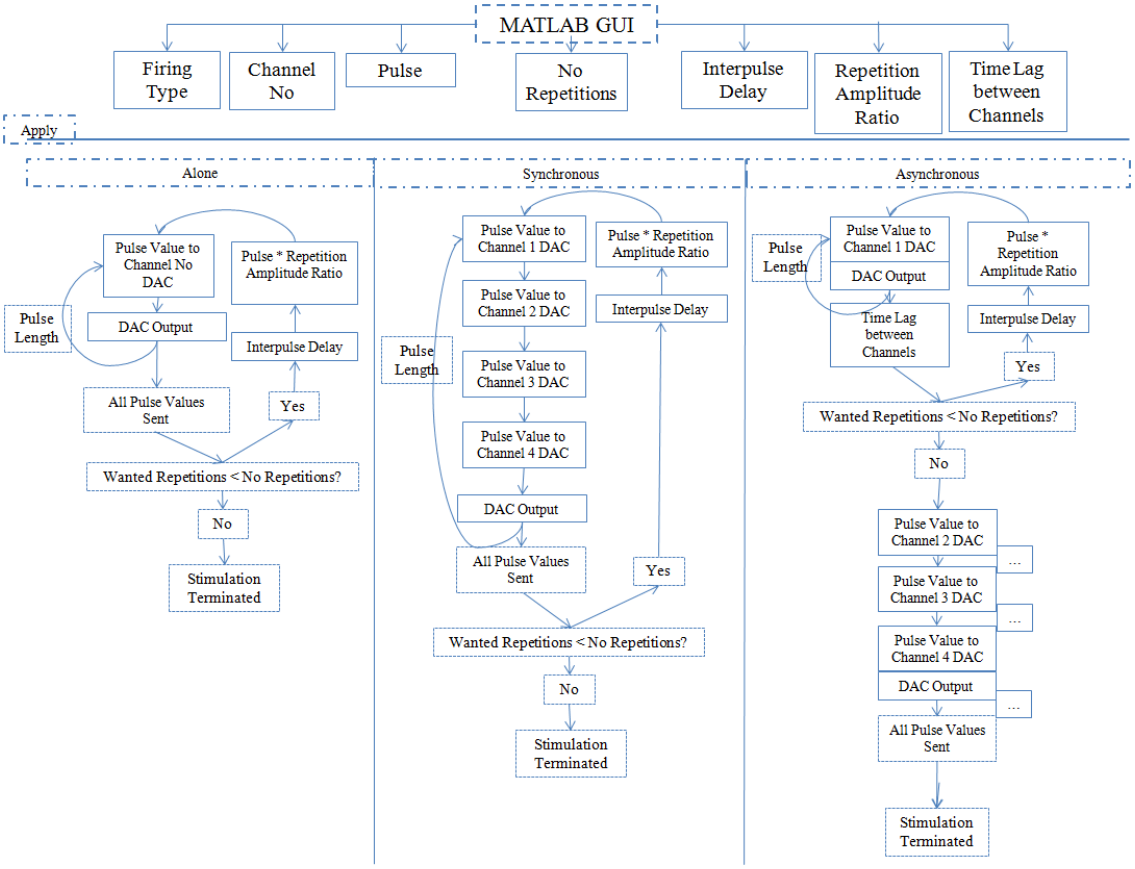

FIGURE 4.31: MCU Operation Diagram.

The microcontroller's role starts when information is being sent by MATLAB. As it is possible to see, the information is: firing type, channel number, single stimulation pulse, number of repetitions, delay between repetitions (interpulse delay), repetition amplitude ratio and time lag between channels.

The first step of the microcontroller is to identify the firing type being used. If it is the alone firing type, then each value from the single stimulation pulse is sent to the DAC via SPI and output by the user-defined DAC channel. After all values from a single pulse are sent, the microcontroller checks if there are repetitions to be done. If not, it ends the program, otherwise it calculates and updates the pulse values, depending on the value of the repetition amplitude ratio, and causes a delay, the interpulse delay, after which it starts sending values to DAC which outputs it. This cycle is repeated until there are no more repetitions to be made.

If the synchronized firing type is chosen, the main difference from the alone firing type is that instead of sending only one value to the DAC, one value per channel used is sent from the respective single stimulation pulse. Only after every used channel has its values, the DAC outputs the values, from all channels at the same time. After all values from a single pulse from all channels are sent and output, the procedure is similar: the microcontroller checks if there are repetitions to be done. If not, it ends the program, otherwise it calculates and updates the pulse values, depending on the value of the repetition amplitude ratio, and causes a delay, the interpulse delay, after which it starts sending values to the DAC which outputs it. This cycle is repeated until there are no more repetitions to be made.

Lastly, if the asynchronous firing type is chosen, the microcontroller essentially produces one alone firing type after the other, one for each of the channels selected, with a delay between them – the time lag between channels. After this the DAC output phase is done. The repetition procedure is the same as previous firing types. Besides the indicated, other information is also received together with the pulse values, as referred to in the previous section, which is the location of signal variations on the pulse. This information is stored along with the pulse and used with it. Its purpose is the activation and deactivation of pins whose function is to control a switch which will allow for change of current direction and thus obtain both cathodic and anodic phases. This is needed because all pulse values stored in the microcontroller are absolute values and also because the DAC used does not output negative voltages. This is further explained in more detail in the following chapter.

It is important to refer that the microcontroller also has LED signaling that is used to indicate its internal state:

- Green LED – no information stored in the microcontroller. This LED turns on when the microcontroller is powered up, after stimulation is applied or if the Reset button is pushed.
- Blue LED – information stored in the microcontroller. This LED turns on after upload is done.
- Red LED – work in process. This can either be when an upload is being done and when stimulation is being applied.

Chapter 5

Hardware Development

As mentioned previously, the stimulation system whose development is being described in this thesis is composed by two main parts, software and hardware. In Chapter 4, a description of the software component of the stimulation system was done. In this chapter we will describe the hardware part of this system. The hardware part of this system is responsible for application of the electrical neural stimulation according to the information that comes from the user configuration done in the software part.

Throughout this chapter a description of the requirements for this hardware platform will be done, the different components that make the platform and the work developed in each of them will be described, and lastly a description of the platform operation will be made.

To design and develop the circuits for the board, simulations of circuits were firstly done through the use of *Cadence* OrCAD Capture and *Texas Instruments* TINA. These are circuit simulators that allow for circuit sketching and simulation using components from external sources, such as the fabricators of the components, thus allowing a more reliable simulation. After this stage the circuits were tested in a breadboard. The actual board was developed using *Altium* Designer program which allows the development of printed circuit boards (PCB) by firstly providing a schematic area where the circuits can be sketched – also with components from external sources – and then, based on those circuits, a 3D workspace is presented where a PCB board appears along with the components defined in the sketch of the circuit. In this 3D workspace the user can then shape the board and place the components in the board where desired. After that the

connections between the components are created, which can be done via auto-routing – where the program tries itself to make the connections – or it can be done by the user. This can be done on the top surface of the board or using any of the layers of the board, a characteristic of a tridimensional board that can also be configured by the user.

5.1 Requirements

As stated before, the hardware part of this system is responsible for the actual application of neural stimulation according to what was configured by the user in the user-system interface. Even though its operation is controlled by the microcontroller, the front-end board should be able to transform that control into a correct stimulation. With this in mind, essential requirements were defined for this board:

- To have its own power supply;
- To have its own voltage reference;
- To be able to receive digital pulse information from MCU and convert it to analog amplitude value;
- To have four equal stimulation channels;
- To apply a current-controlled stimulation;
- To be able to apply bi-directional current.

The following chapters will describe the components chosen and the work developed to accomplish all these requirements for the stimulation system front-end board.

5.2 Components

In order to accomplish a current-controlled stimulation system, the front-end board was defined to be composed by power supply circuits, reference voltage circuits, a 16-bit quad-DAC, opamp-based voltage-to-current converter and a set of switches. In the following sections a description of the referred board components will be done, focusing on

the work developed in each of them and their role in the operation of the board. Figure 5.1 shows an *Altium* Designer 3D vision of the developed board with the referred circuit components highlighted and identified.

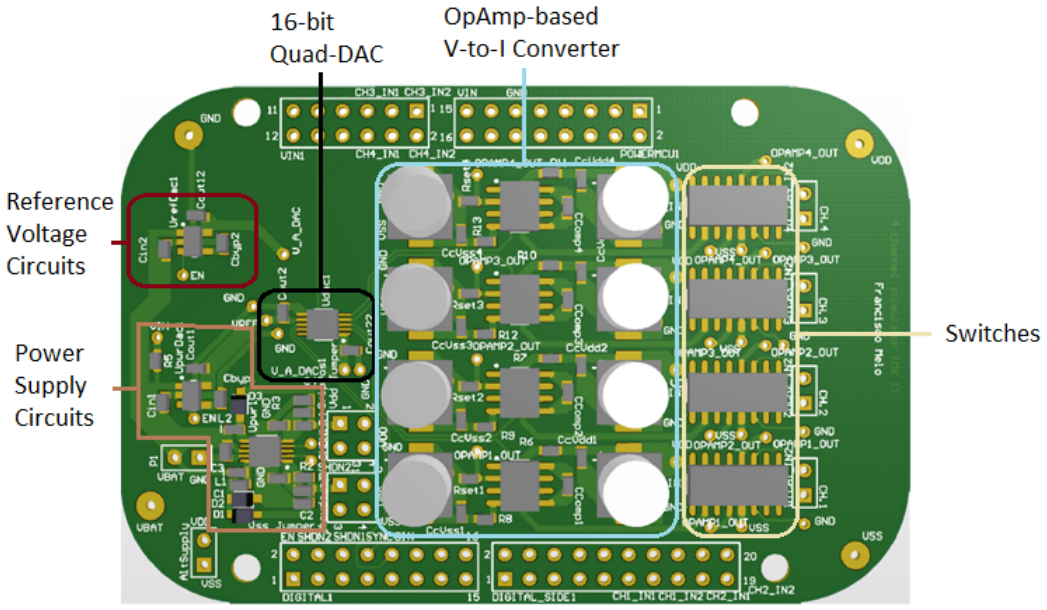

FIGURE 5.1: *Altium* 3D vision from the board with highlighted components

5.2.1 Power Supply Circuits

Identified by the brown line in Figure 5.1 are the power supply circuits. As previously mentioned, it was a requirement that this board could have its own power supply, distinct from the MCU supply source, which is powered by the USB connection to the computer. To do so, a power supply circuit had to be designed so that power could come from an independent external source and be steady and properly delivered to the electronic parts. These circuits were then designed to meet the requirements from both the parts that need the power supply, namely the DAC, operational amplifiers (opamps) and switches, and also of the parts used to obtain the desired voltage for supply from the external source voltage, namely the DC-to-DC (DC/DC) Converter and Low-Dropout Regulator (LDO). A DC/DC converter is an electronic converter which has the capacity of turning a DC voltage source into a DC output of different voltage value. However to work properly it usually needs to be used together with other electronic parts. In Figure 5.2, the LT1945 DC/DC converter circuit for positive and negative power supply schematic is presented. The LT1945 was chosen not only for its capacity to provide both

negative and positive output but also because it can do it from a single positive supply with a range from 1.2 V to 15 V. This circuit, and the values of additional parts used, is based on recommendations from the datasheet of DC/DC converter used in order to obtain an output in negative supply (VSS) of -9 V and in positive supply (VDD) of +9 V and it was successfully tested with a external supply (VBAT) of 5.5 V.

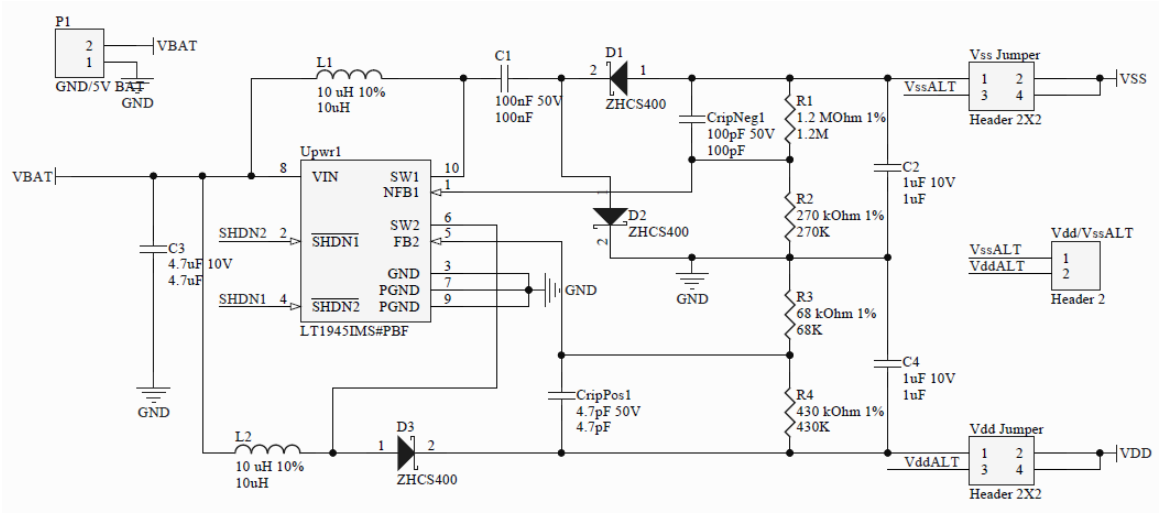

FIGURE 5.2: DC/DC converter circuit schematic.

As it is possible to see in the schematics presented in Figure 5.2, four headers were used in this circuit. The first one, 'P1', is the one where the external power supply should be connected to power up the DC/DC converter. At the end of the DC/DC converter circuit there are 2 headers, 'Vss_Jumper' and 'Vdd_Jumper', which are here placed with the purpose of allowing the user to choose between using the DC/DC converter supply to the components or a different external supply. The 'Vdd/VssALT' header is then to allow the user to connect the external supply. The placements in the board are shown in Figure 5.3 and highlighted by the black, blue and red circle, respectively.

These defined positive and negative values output by the DC/DC converter circuit are suitable for supply of most electronic parts used in this board except for the DAC which, in turn, needs a supply voltage of 5 V. In this case, the voltage supply is given by the LP3985 LDO voltage regulator whose schematic is in Figure 5.4. A LDO regulator is a linear voltage regulator which can have a steady output voltage from a defined range of input voltage and whose difference between input and output can be small – low dropout. This LDO in particular, the LP3985, was chosen because it is low power, has low noise and very low-dropout voltage and it can have a 5 V output from an input

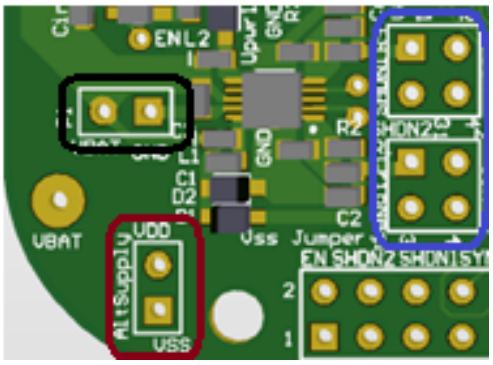

FIGURE 5.3: Placement on the board of VBAT power supply header (black), VDD and VSS jumpers (blue) and Vdd/VssALT alternative supply header (red).

range of 5 V plus dropout voltage, which was of 100 mV, up to 6 V. This input range is included in the range of DC/DC converter input which is important for the definition of external source voltage.

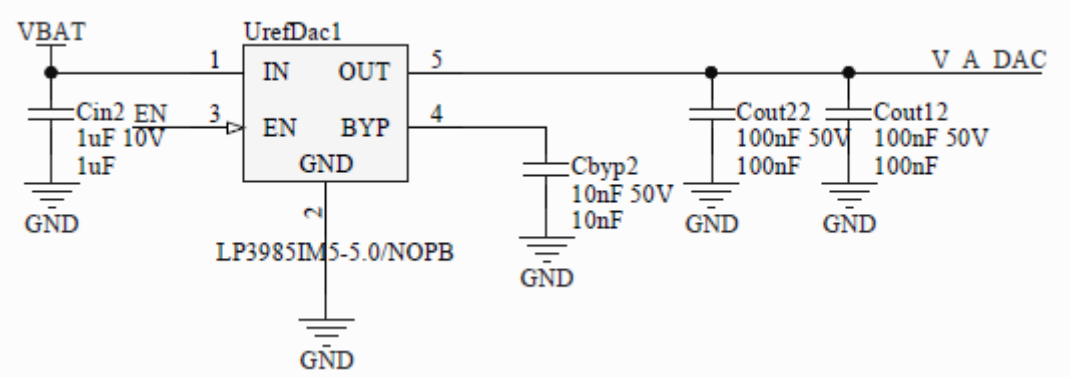

FIGURE 5.4: Schematic of DAC's power supply circuit.

Other important characteristic of the DC/DC converter and LDO chosen is the possibility of shutting down the output until desired through an enabling pin. In the DC/DC converter, there are two enabling pins, identified as SHDN1 and SHDN2, and in the LDO there is one, identified as EN. These are logical pins which enable output when they are high and disable when low and are connected to the MCU which is programmed to only enable them when MCU is completely functional. This way, by controlling these pins, there is no need to wait for the MCU to be on to connect the board to the battery to avoid any unintentional charge delivery.

It is important to refer that a misnaming happened in the designing of the board. In Figure 5.4, the name of the component should be 'UpwrDac1' instead of 'UrefDac1', and

the inverse problem is verified in the reference circuit for the DAC.

5.2.2 Reference Voltage Circuits

Identified by the red line in Figure 5.1 is the reference voltage circuit. As previously mentioned, it was a requirement that this board could have its own reference voltage, distinct from the power supply circuits. This circuit is equal to the one responsible for the DAC power supply, with the same source from the battery and an output of 5 V as well, as this was defined as the voltage reference value. DAC's power supply and reference voltage could be connected to the same LDO output however it was decided to have one for each so that higher stability could be obtained. Figure 5.5 shows the schematic of the circuit. It is important to remind, once again, of the misnaming of LDO in the schematic and in the board.

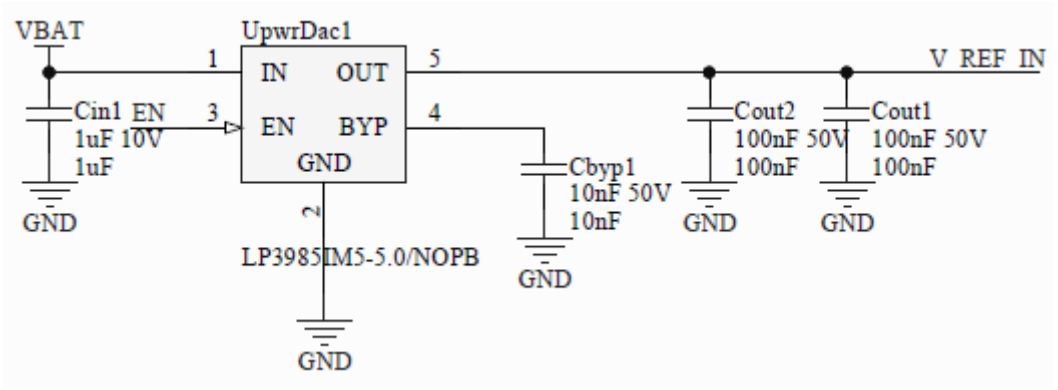

FIGURE 5.5: Schematic of DAC's reference voltage circuit.

5.2.3 Stimulation Application Circuits

Highlighted by the black, blue and white lines in Figure 5.1 are, respectively, the DAC, voltage-to-current (V-to-I) converter and switches. As initially mentioned, it was a requirement that this board would be able to receive digital pulse information from MCU and convert it to analog amplitude value and also that would have four equal stimulation channels. To accomplish this, there were two options: either to have four DAC, one DAC for each channel, or a DAC with four outputs, called quad-DAC. For this board the second option was decided, with the DAC124S085 Quad-DAC being the chosen one. The choice of this DAC is due to the fact that it is based in a 16-bit serial communication,

which is the same communication type of MCU and also because, of the sixteen bits, twelve are for the digital values, meaning it allows to have a good amplitude resolution – approximately 1.2 mV for this case since the DAC can output a voltage amplitude up to voltage reference which is 5 V – and the remaining four bits are for channel selection and mode of operation, which include important options such as sending information to a specified DAC channel and output it or not, or to send an information to all the channels and output it. However, when output is done, all channels give an output and it is not possible to have only one at a time. The connections of DAC used are represented in the schematic of Figure 5.6.

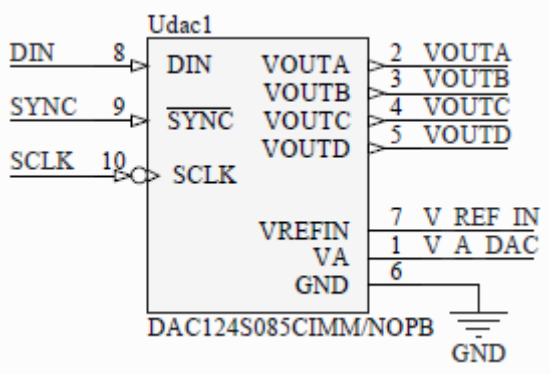

FIGURE 5.6: DAC schematic representation.

After the DAC, is then a V-to-I converter followed by a switch. The V-to-I converter is necessary because it is a requirement to be a current-controlled stimulation system and the DAC output is voltage amplitude. It was desired that this V-to-I converter could be as simple as possible so it was initially considered to use the *Texas Instruments* XTR111 V-to-I converter, however this product was only available as Surface Mount Device (SMD) and so to test it in a breadboard would mean that other electronic parts had also to be bought making this an expensive option. The least expensive alternative thought was then to design our own V-to-I converter based on an opamp, this way allowing for easier and cheaper testing on a breadboard. To maintain as simple as possible, the circuit designed was simply an opamp-based V-to-I converter, as it is presented in Figure 5.7. The opamp chosen was the AD744 due to its fast-settling, relatively good slew-rate, low total harmonic distortion (THD) and external compensation for stable operation.

In a V-to-I converter such as the one in Figure 5.7, the current amplitude generated by

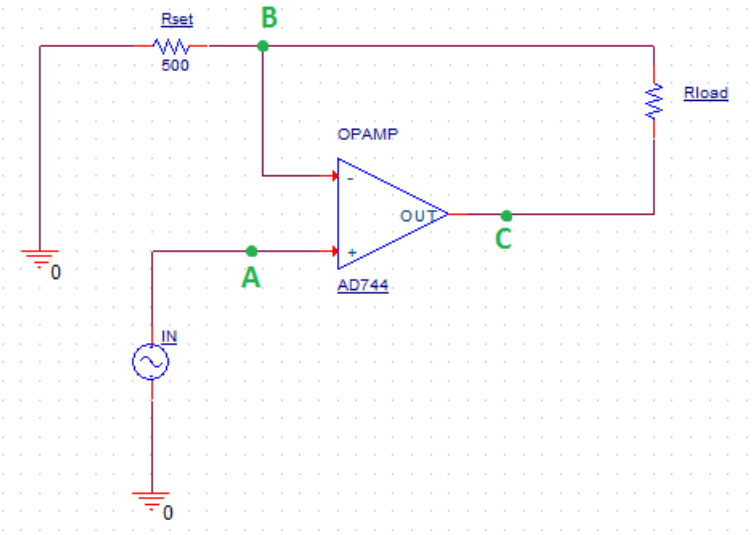

FIGURE 5.7: *Cadence* OrCAD schematic of V-to-I Converter.

the input voltage amplitude is defined by the setting resistor R_{set}, since the voltage potential in inverting input V_B is the same as in non-inverting input V_A, which corresponds to the input voltage coming from the DAC, and R_{set} is between V_B and ground. As current entering the opamp input is substantial, the current in R_{set} is the same in R_{load} which in this situation represents the impedance from switches, electrodes and tissue. However, this also means that the voltage potential on the output of the opamp, V_C, is dependent on the value of R_{load} and this simplicity of the V-to-I converter circuit leads to a major limitation which is related to the maximum amplitude that can be at opamp output, i.e., the opamp saturation limits which are given by the power supply. This means that R_{set} had to be chosen carefully so that it could have a good current range but could also handle reasonable values of load impedance. The maximum current amplitude was defined as 10 mA with a 500 Ohm setting resistor, since the maximum DAC output is 5 V. This limitation here referred is the reason why in user-system interface there is maximum voltage amplitude warning. As tissue impedance is very small compared to electrode it is neglected and only switch and electrode impedance is considered as load impedance in calculation of maximum amplitude voltage – see Equation 4.8 in Software Development chapter.

By its turn, the switch is necessary to allow for bi-directional current as required, namely cathodic and anodic current which are very important for charge-balancing. The switch used is an ADG436 Dual Single Pole, Double Throw (SPDT) Switch and its scheme is presented in Figure 5.8. The scheme of this switch then allows designing the circuit so

that not only switching the direction of the current going through the electrodes can be done but also there is a conformation where no current is directed to the tissue.

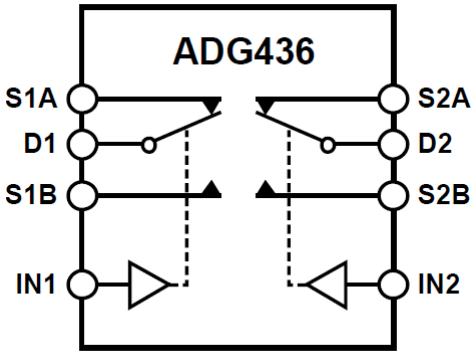

FIGURE 5.8: ADG436 switch scheme.

In this switch, when its Logic Control Input (INx) is low the Dx pin is connected to SxB, when INx is high Dx pin is connected to SxA. The schematic of the circuit designed is presented in Figure 5.9. This means that when IN1 is low and IN2 is high, and so S1A is connected to D1 and D2 is connected to S2B, cathodic current is obtained. When IN1 is high and IN2 is low, and so S1B is connected to D1 and D2 is connected to S2A, the opposite current, anodic, is obtained. When both IN1 and IN2 are low, D1 is connected to S1B and D2 is connected to S2B, and there is no feedback circuit which leads to absence of current going to the tissue and may also allow some remaining charge on the tissue to be dissipated.

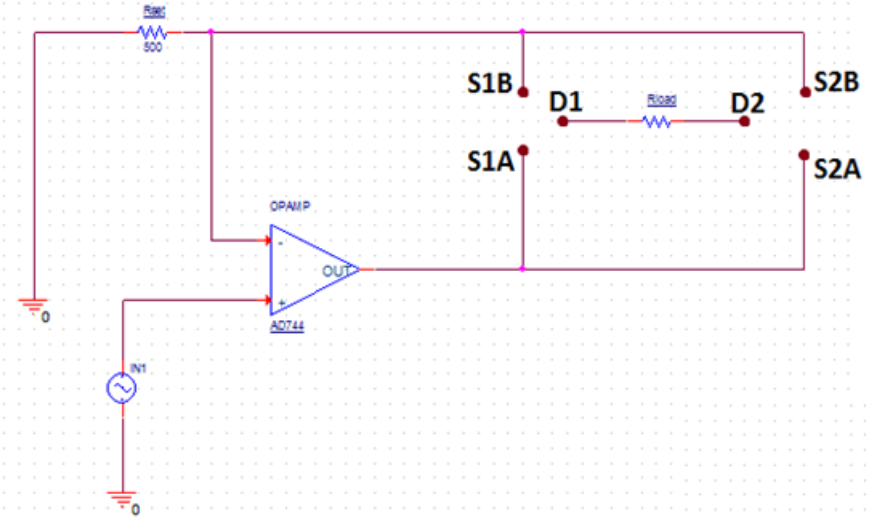

FIGURE 5.9: Schematic of circuit with V-to-I Converter and Switch – single stimulation channel.

In Figure 5.10 is shown the full schematic of a single current-controlled stimulation channel, with both the opamp-based V-to-I converter and switch, in which are included the impedance from the switch – R_{swtA}, C_{stwA} from switch bridge A and R_{swtB}, C_{swtB} from switch bridge B – and also the impedance from the electrode and tissue as described by the Electrode-Electrolyte Model – R_{farad1}, C_{dl1} from electrode 1, R_s of electrolyte or tissue resistance, and R_{farad2}, C_{dl2} from electrode 2.

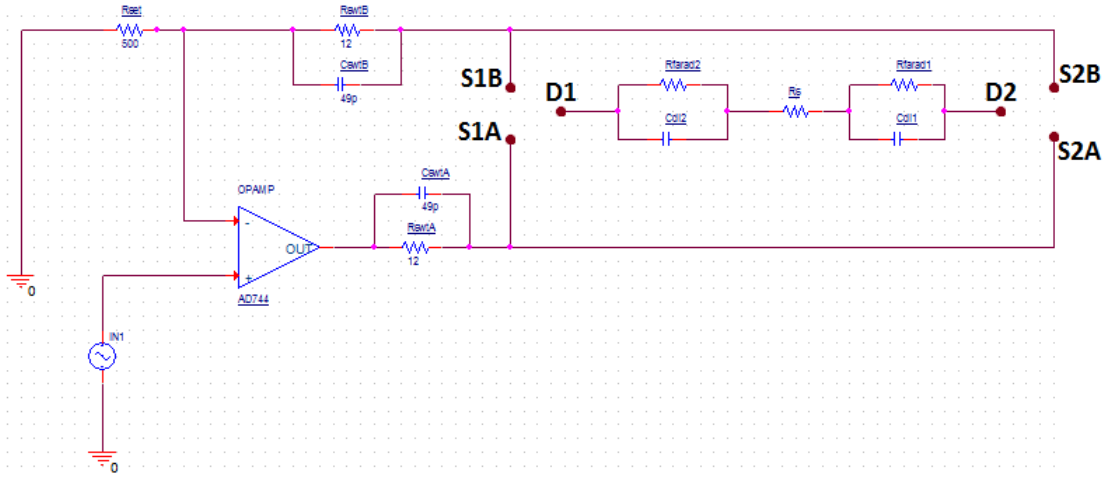

FIGURE 5.10: Schematic of a single stimulation channel circuit with converter, switches and electrode impedances.

In Figure 5.11 is shown the same single stimulation channel but in the Altium schematic.

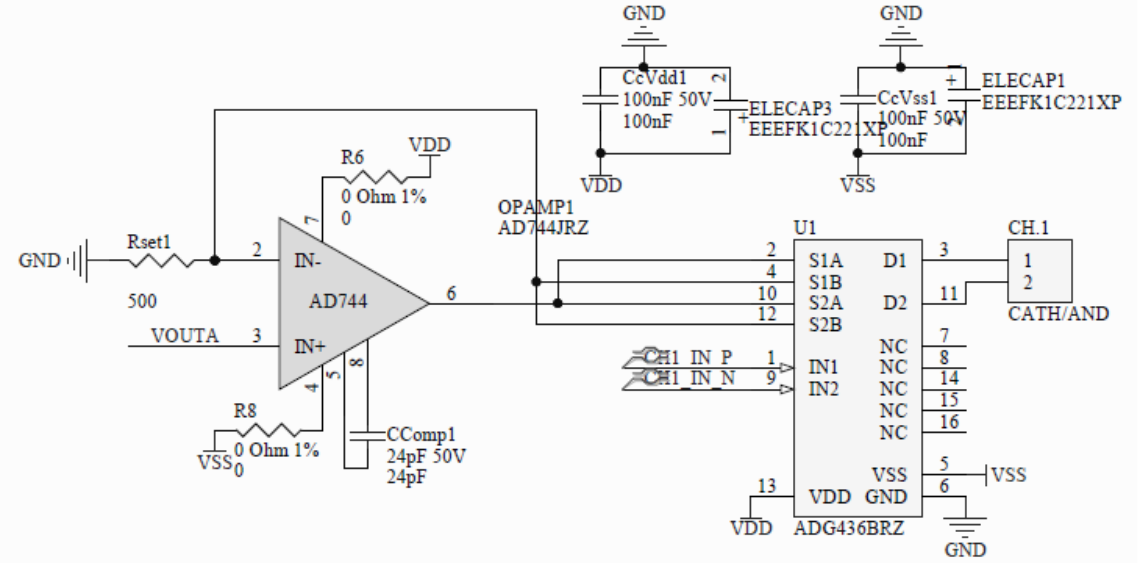

FIGURE 5.11: *Altium* schematic of a single stimulation channel circuit.

On the top right of the schematic are represented the decoupling capacitors for the VDD and VSS sources. On the left is then the V-to-I converter and on the right, under the

decoupling capacitors, is the switch. Zero Ohm resistors were also placed between the opamp power connections and the power sources in order to allow easy disconnection of a channel and board debugging. It is also possible to see in the schematic the header were the electrodes should be connected, called CH.x, where x will be the number of the channel (1, 2, 3 or 4), which is the place where current will be output from the board to the stimulation site. Their placement on the board is shown in Figure 5.12.

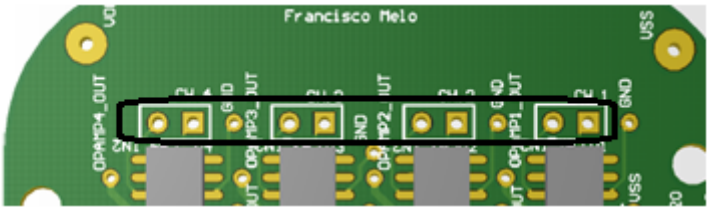

FIGURE 5.12: Electrode placement sites on board.

5.2.4 Microcontroller Connections

The front-end stimulation board here developed is, as it has been mentioned throughout the thesis, essentially controlled by the FDRM-KL26Z microcontroller. To accomplish that control, sites for input and output connections had to be created to allow connection to the specific components. Figure 5.13 presents a 3D vision of the whole stimulation board created with these connections properly identified.

Identified on the Figure 5.13 are then the following board connections to the KL26Z:

- CHx_INxk – is the point of connection between microcontroller and the switch's logic control input.
- DIN – DAC's Serial Data Input.
- SYNC – DAC's Synchronization Input for data input.
- SHDNx – Enable pins (also, shutdown pins) for DC/DC converter.
- UART_TX – connection point to UART transmitter.
- UART_RX – connection point for UART receiver.
- EN – Enable pin for LDOs.
- SCLK – DAC's Serial Clock Input.

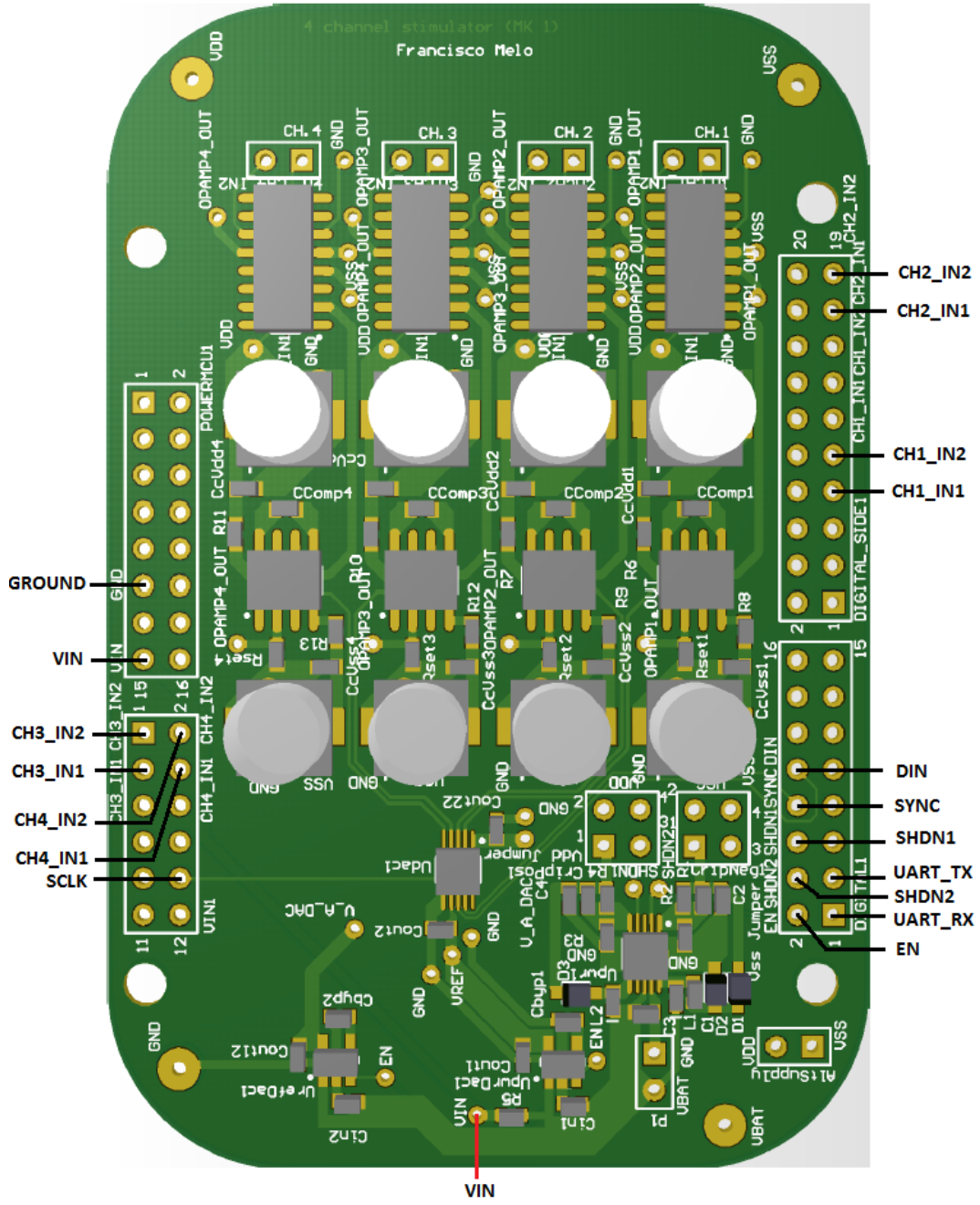

FIGURE 5.13: Input and Output Connection on whole Stimulation Board.

- GROUND – provides ground to the whole system.
- VIN – used to power the microcontroller if it is not connected via USB. In this case, a wire has to be used to connect the connection pin VIN (identified in black) to the power pin (identified in red).

5.3 PCB Design Characteristics

Besides the component placement on the PCB board, there are other very important characteristics that have to be defined to make a perfectly operational board. These characteristics are the connections between the components and the board layers.

As previously mentioned, the connections can be done via auto-routing or can be done by the user. In auto-routing mode the program tries itself to make the connections which, although fast and convenient, have significant limitations. A particular and very important limitation is related to the lack of knowledge of the program regarding the type and amplitude of signals that will pass in each of the connections which is an important concern in order to avoid interference and contamination of signals, also called signal cross-talk. This can occur for example when a wire carrying an analog signal passes near a track carrying a digital signal. It can also occur for example when a high amplitude signal passes near a lower amplitude signal. Another important concern is the width of the tracks carrying the signal, where the greater the signal current intensity, the wider the track should be. This relation comes from the fact that current passing causes a temperature rise on the track which increases with the increasing of the current intensity. This way a wider track allows a better heat dissipation, keeping the temperature within safe and less-damaging limits. For all these reasons, the connections were made by hand.

In Figure 5.14, the connections made between all the components on the board, including the input and output sites, are shown. As it is possible to see, there are tracks that are much wider than the others. These tracks are the ones related to the power supplies circuits since these are the circuits that are required to conduct higher currents. Following these are the voltage reference circuits, which have slightly wider tracks than the remaining connections.

It is also possible to see in Figure 5.14 that the tracks have different colors: red, brown, light blue and dark blue. This color difference is related to the layer where the tracks were made. The red tracks are on the top layer, the brown tracks are on a top-internal layer, the light blue tracks are on a bottom-internal layer, and the dark blue tracks are on the bottom layer. The role of each layer is shown in Figure 5.15 and, as it is indicated, the top layer is mainly for analog signals, the top-internal layer for digital signals, the bottom-internal layer for power connections and the bottom layer for ground

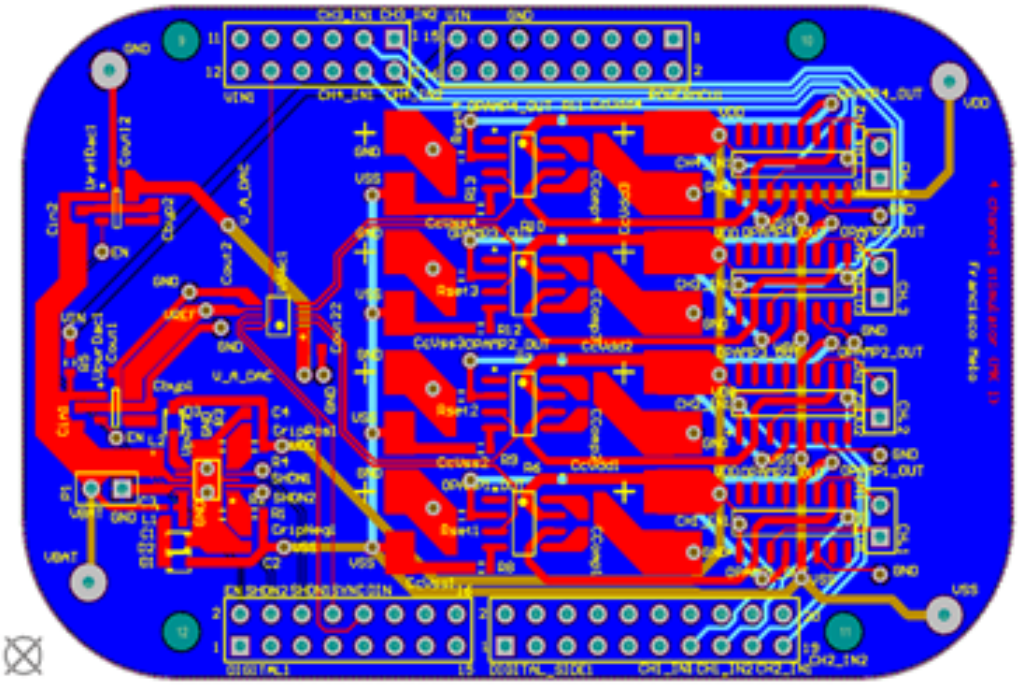

FIGURE 5.14: Board Connections on *Altium*.

connections. The possibility of multilayer routing not only facilitates the hand routing, by allowing that tracks can pass through the same point, bi-dimensionally speaking, but also because it provides isolation to avoid cross-talk between the tracks. Figure 5.15 also shows how the communication between layers is made in a multilayer routing. As it is possible to see, the communication is made through holes that cross the entire board, called "through-holes", which then connect to a defined layer. In this example, the left 'through-hole' allows for connection between a top-layer track and top-internal layer, without connecting with either the bottom-internal or the bottom-layer track. On the 'through-hole' on the right however, the top-layer track is connected to both the bottom-internal and the bottom-layer track, but not to the top-internal layer. In between these layers there are non-conductive layers to isolate the conductive ones.

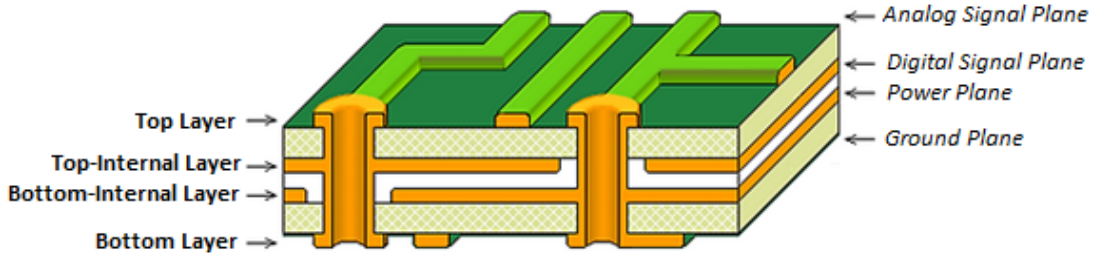

FIGURE 5.15: Layers connections.

It is also important to refer that a pouring layer was placed in the bottom layer, covering the free area of this layer, thus the reason why the board looks dark blue making it hard to understand where the bottom-layer tracks are. The reason to include such a layer is related to the expected use of this board, i.e., since the board is intended to be used with the microcontroller, being installed on the top of it, this pouring layer then acts as a protective layer, between the board and the microcontroller, to avoid interferences between them.

5.4 Full Neural Stimulation Board and MCU Compatibility

The result of the front-end stimulation board here developed is shown in Figures 5.16, 5.17 and 5.18. In Figure 5.16 it is possible to see a 3D image of the full stimulation board with all its components. In Figure 5.17 the real version of the stimulation board is shown only with the tracks and connections for the components. Finally, in Figure 5.18, the full stimulation system with all components already placed and soldered to the board is shown.

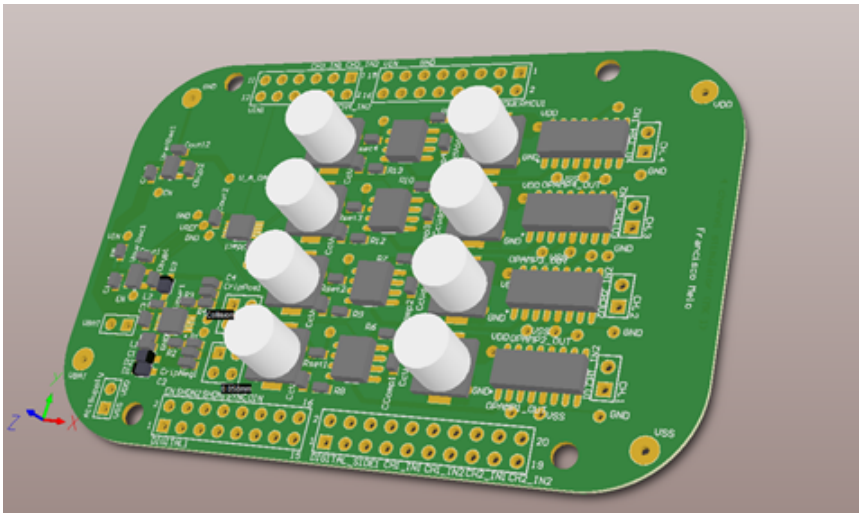

FIGURE 5.16: 3D image of Stimulation Board.

Figure 5.19 shows an image of FRDM-KL26Z microcontroller with its input/output connectors highlighted. For each connector, their connection to the inputs of the front-end stimulation board are identified.

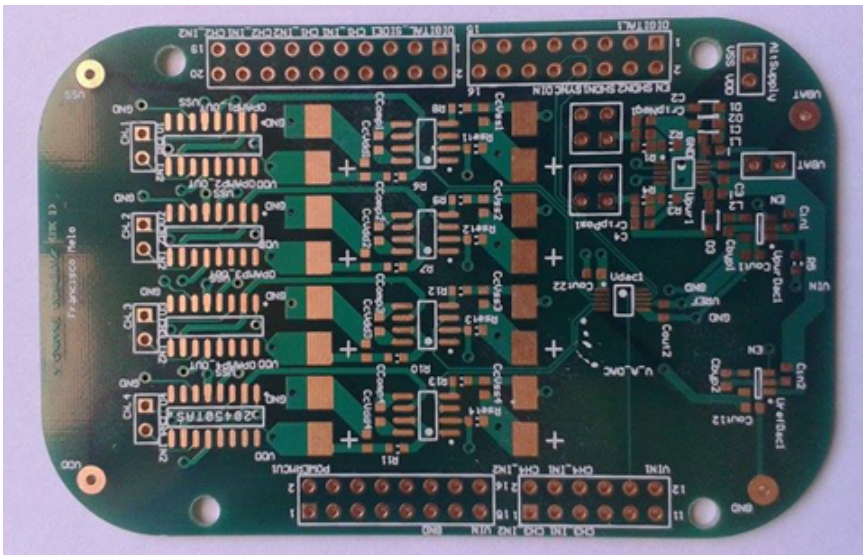

FIGURE 5.17: Tracks in the real board without any components.

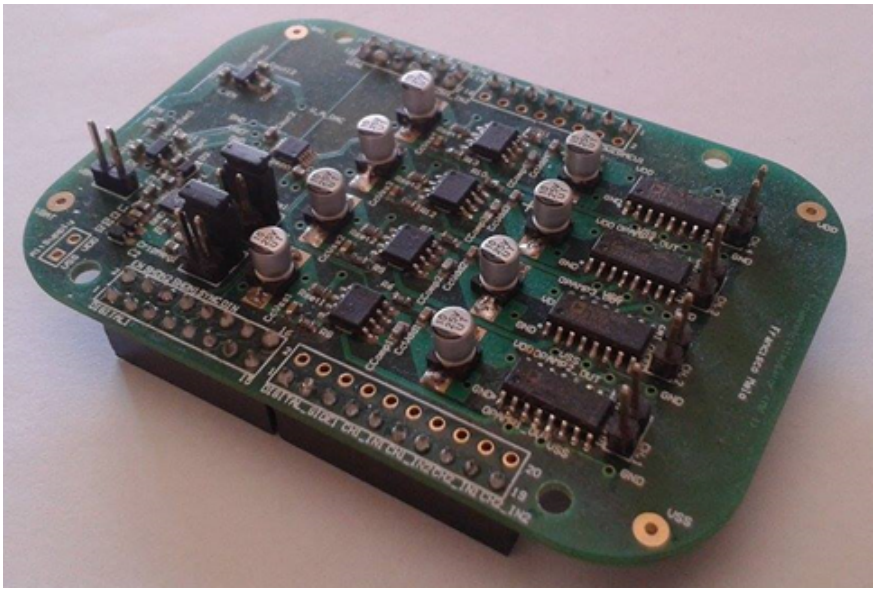

FIGURE 5.18: Full Stimulation Board, with all components.

It is important to refer that this system was built with the future plan of being used together with a neural recording system with a structure and control similar to this stimulation board. So, when defining the connections of the MCU to the stimulation board, attention had to be given to the connections that would be required by the recording system. This way, overlapping of connectors would not occur, except for connections in which communication could be made through the same inputs without affecting the functioning of the connections of either the stimulation or the recording systems to the microcontroller. These connectors are the UART connection pins, the LED connection and the MCU power supply pin.

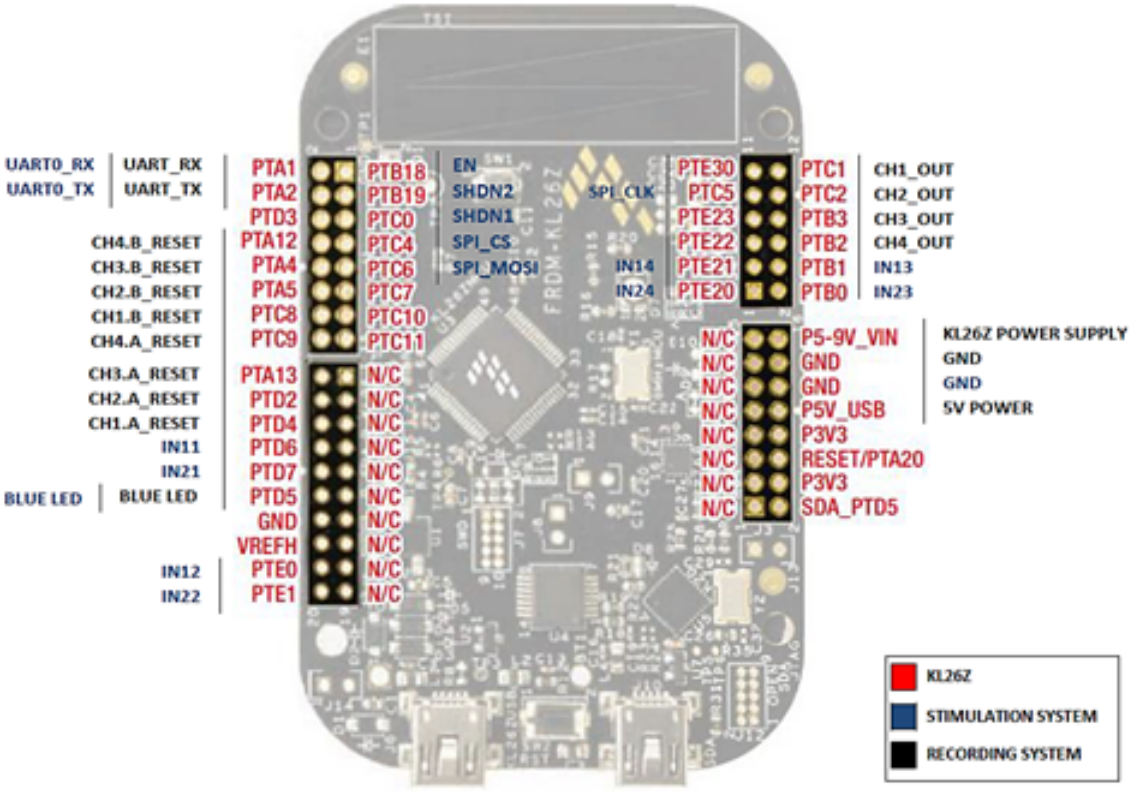

FIGURE 5.19: FRDM-KL26Z connections with Stimulation and Recording Board.

Chapter 6

Overall Stimulation System Operation

Throughout this thesis, the three main components of an embedded platform for electrical neural stimulation were described, namely the User-System Interface, the Microcontroller – Chapter 4 – and the Front-End Board – Chapter 5. The result of the developed work was an operational current-controlled stimulation system whose stimulation profiles are configurable by the created user-system interface. Figure 6.1 shows the overall operation of this stimulation system.

FIGURE 6.1: Overall Operation of the Embedded Platform for Neural Electrical Stimulation.

As it is possible to see in Figure 6.1, the operation of the system starts with the user programming the stimulation profiles from which results the stimulation information, such as one pulse stimulation and other parameters. That information is then sent to the FDRM-KL26Z microcontroller via the USB connection when the user presses 'Upload' in the interface. The microcontroller then stores that information and starts to process

it when 'Apply' is pressed. The processing in the microcontroller consists of controlling the Front-End board in accordance with the stored information from the interface. This includes sending pulse values to the DAC, controlling its output according to information on firing type and stimulation channel, and controlling the switches according to the phase of the pulse. The end of the operation resulting from a stimulation profile uploaded and applied is then at the Front-End board, which is responsible for converting the digital pulse value coming from microcontroller into voltage amplitude, with the DAC. The voltage is then converted into current, via the voltage-to-current converter, which is then routed to the switch, whose conformation depends of pulse phase and is finally output through one of the two electrodes and recollected by the other one.

6.1 System Analysis

To check the operation of the built stimulation system, tests were made with the use of an oscilloscope. The tests consisted essentially in applying the various possible types of stimulation profiles and measuring the output on the electrode sites of the front-end board. Figures 6.2, 6.3, 6.4, 6.5 and 6.6 show, respectively, the output of the board when a single stimulation pulse is done with a profile based on a rectangular waveform, bi-level Lilly waveform, quasi-trapezoidal waveform, tri-exponential waveform and bursting strategies. All waveforms were configured to deliver a 850 mV primary amplitude and a secondary amplitude of 425 mV, when applicable, with the remaining parameters left with the preset values reffered in Chapter 4. These signals were measured between the entry of a dummy load composed by a 150 Ohm resistor and the anodic electrode site of Channel 1. The reason for choosing this point for the measures was the fact that, since charge-balancing values are smaller, the differences between the expected value and the measured one, resulting from the impedance of the dummy load, would be smaller as well. This would allow for a better evaluation of the similarity between the stimulation and what was configured. These figures show not only the voltage measured at the dummy load entry, in yellow, but also the voltage at the input entry of the switch, in pink. The latter is up for anodic current, so that it could be easier to see the low amplitude in anode phase. In these testing conditions it was expected to obtain, for the rectangular shaped stimulation, a charge voltage amplitude of 850 mV and a charge-balancing amplitude of 110 mV. The latter amplitude is slightly higher than the calculated 85 mV since in the

charge-balancing conformation the anodic site is the opamp output. Values similar to the ones expected were obtained, as it can be seen in Figure 6.2.

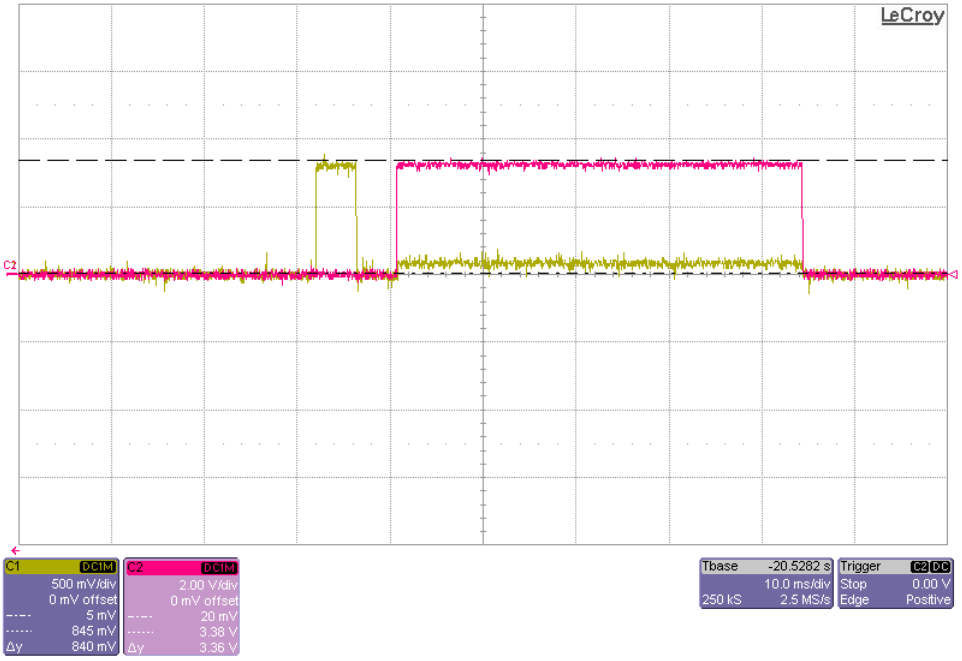

FIGURE 6.2: Measured wave for rectangular shaped stimulation.

For the bi-level Lilly shaped stimulation it was expected to obtain a primary charge voltage amplitude of 850 mV, a secondary amplitude of 425 mV and a charge-balancing amplitude of 164 mV. The latter was, again, slightly higher than the calculated value of 127.5 mV, for the same reason explained before. Values similar to the expected ones were obtained as it can be seen in Figure 6.3.

For the quasi-trapezoidal shaped stimulation it was expected to obtain a primary charge voltage amplitude of 850 mV and a charge-balancing amplitude slightly higher than the 110.8 mV: 142.7 mV. Values similar to the expected ones were obtained as it can be seen in Figure 6.4.

For the tri-exponential shaped stimulation it was expected to obtain a primary charge voltage amplitude of 850 mV and a charge-balancing amplitude slightly higher than the 28.5 mV: 38 mV. In this case, as it can be seen in Figure 6.5, the charge-balancing values obtained were also similar to the ones expected. The charge values, however, were smaller than the expected.

For the "bursting strategies" based stimulation it was expected to obtain a primary charge voltage amplitude of 850 mV and a charge-balancing amplitude slightly higher

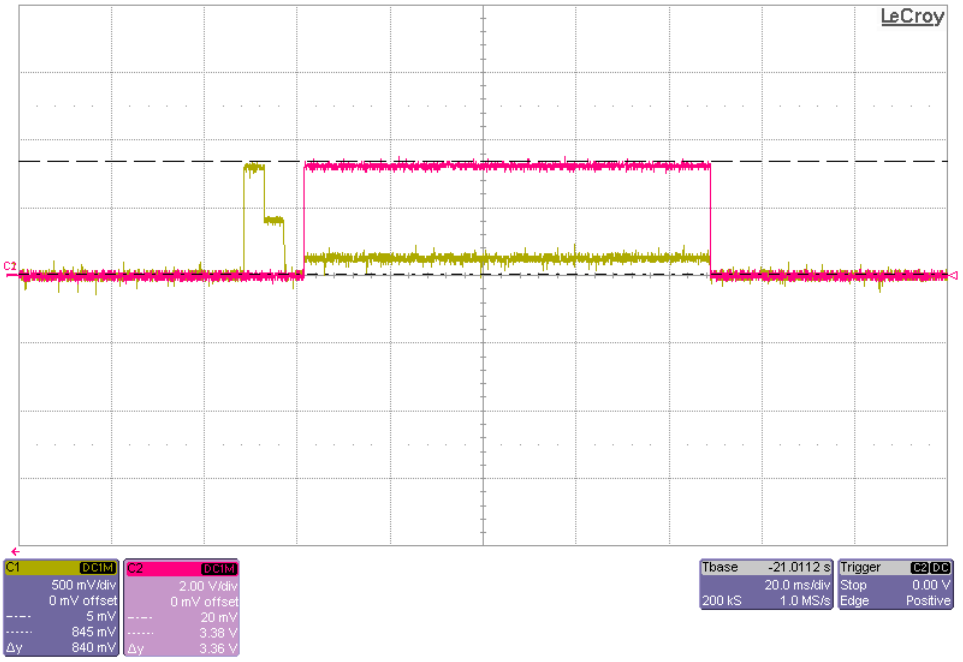

FIGURE 6.3: Measured wave for bi-level Lilly shaped stimulation.

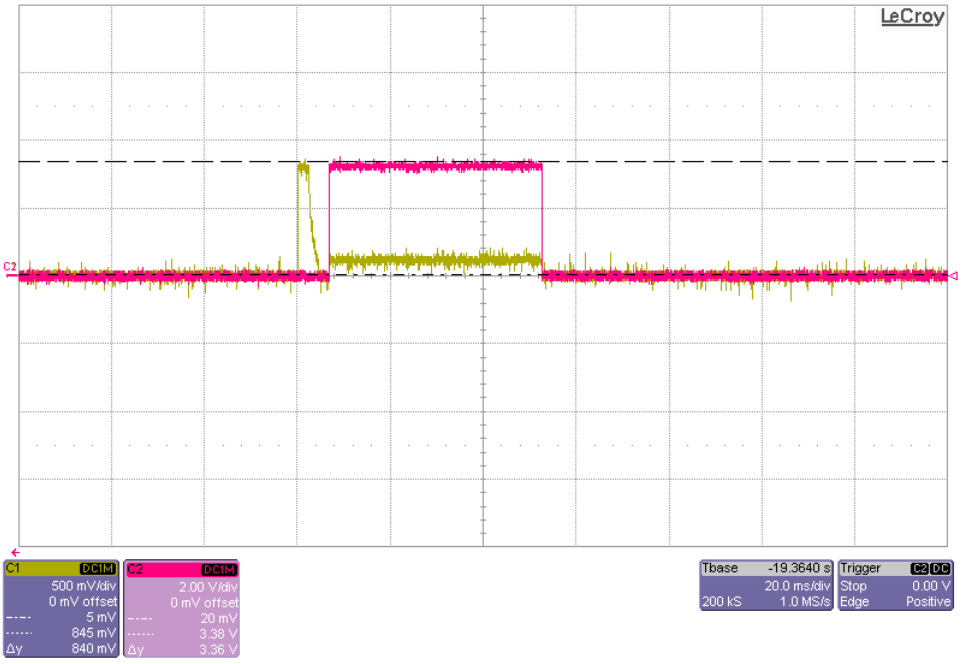

FIGURE 6.4: Measured wave for quasi-trapezoidal shaped stimulation

than the 85 mV: 110 mV. Here, the values obtained were once again similar to the ones expected as it can be seen in Figure 6.6.

Table 6.1 summarizes the results of obtained from previous measures. It can be seen the charge amplitude programmed in the user-system interface and resulting charge-balancing amplitude, the expected charge-balancing amplitude due to the measuring

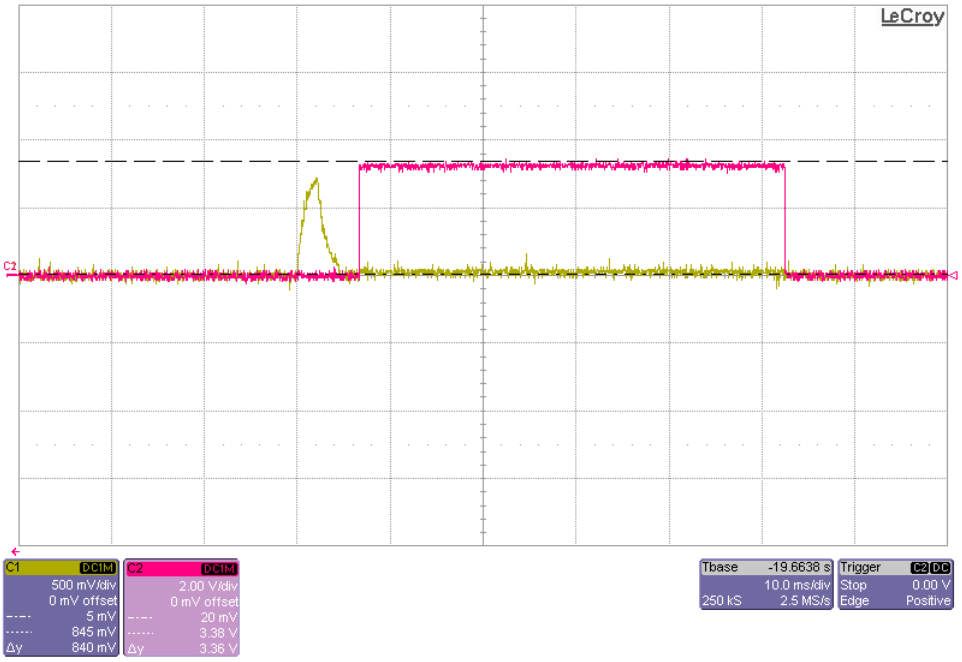

FIGURE 6.5: Measured wave tri-exponential shaped stimulation.

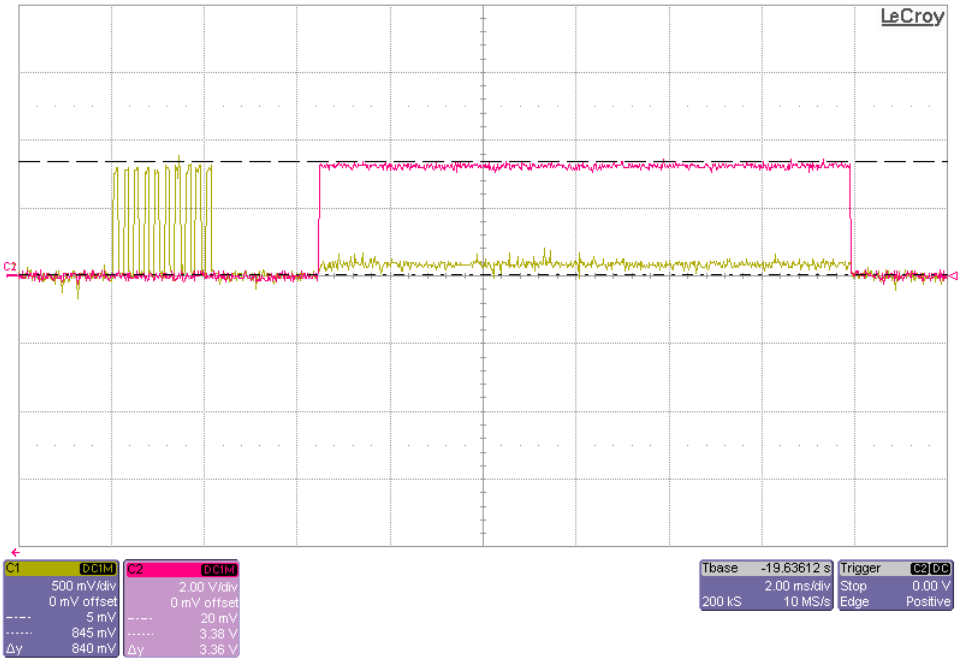

FIGURE 6.6: Measured wave for bursting strategies based stimulation.

conditions, and the approximate measurement values for charge and charge-balancing. Bi-level Lilly presents two values in charge amplitude, the first for the first step amplitude and the second for the second step amplitude. The referred proximity between the values expected and measured can be clearly observed, excepted for the Tri-Exponential waveform, where a slight difference between charge amplitude programmed and measured can be seen.

TABLE 6.1: System Analysis measurements.

Waveform Type	**Programmed Charge Amplitude [mV]**	**Programmed Charge-Balancing Amplitude [mV]**	**Expected Charge-Balancing Amplitude [mV]**	**Measured Charge Amplitude [mV]**	**Measured Charge-Balancing Amplitude [mV]**
Rectangular	850	85	110	840	100
Bi-level Lilly	850[1] 420[2]	127.5	164	840[1] 400[2]	150
Quasi-Trapezoidal	850	110.8	142.7	840	130
Tri-Exponential	850	28.5	38	700	30
Bursting Strategy	850	85	110	840	100

[1] *First step amplitude.* [2] *Second step amplitude.*

The same procedure was also done for the remaining channels and the same results were obtained. Besides single pulse stimulations, the other possible stimulation configurations were also tested. Figure 6.7 shows a rectangular-based repetitive stimulation – in yellow – with a primary amplitude of 700 mV, a repetiton rate of 10 kHz, a repetition amplitude ratio of 2 and a stimulation duration of 40 ms. The blue and pink traces represent the cathodic and anodic phases, respectively, since they are sampling the corresponding switch inputs. The remaining configuration were left as preset.

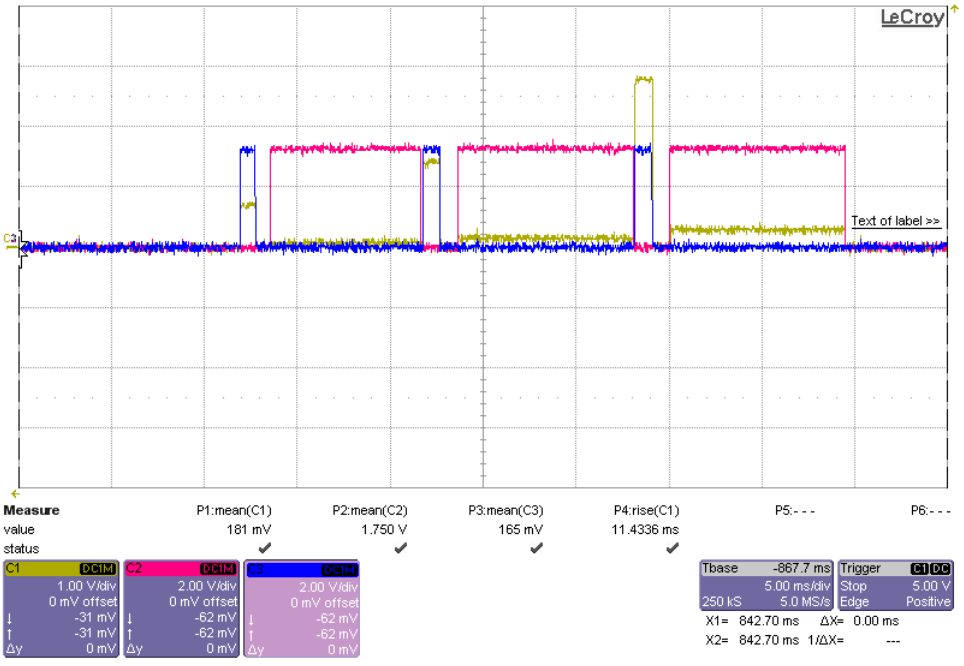

FIGURE 6.7: Rectangular shaped repetitive stimulation.

Figure 6.8 shows a similar stimulation but where, along the repetition, the maximum amplitude voltage is reached as it is possible to understand by the fact that in the last two pulses the amplitude does not increase. To obtain a stimulation as the one presented in Figure 6.8, the following parameters were chosen in the interface: rectangular-based stimulation with a primary amplitude of 100 mV, a repetition rate of 10 kHz, a repetition amplitude ratio of 2 and a stimulation duration of 90ms. All the remaining parameters were left as preset. Also, to test the systems capacity to comply with the calculated voltage maximum, it was configured in the interface that the electrode impedance would be of 900 Ohm, which leads to a voltage maximum of approximately 3.2 V. The result is shown in Figure 6.8 with a stimulation composed by seven repetitions with increasingly higher amplitude until the sixth one, from where the amplitude does not increase more due to the amplitude limitations.

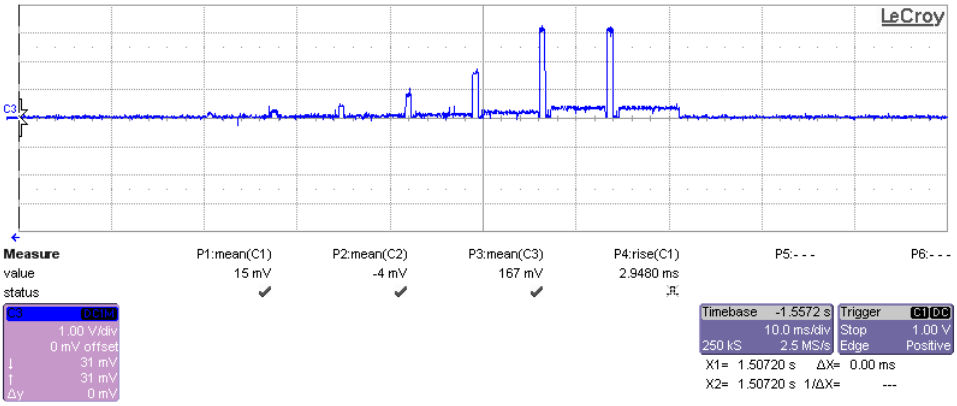

FIGURE 6.8: Rectangular shaped repetitive stimulation with amplitude limitation.

Figure 6.9 shows a synchronous rectangular-based stimulation through two different channels, namely Channel 3 – blue tracing – and Channel 4 – yellow tracing , both with a primary amplitude of 700 mV and the remaining parameters left as preset.

Figure 6.10 shows an asynchronous rectangular-based stimulation through the same two channels, Channel 3 and 4, the first represented by the blue tracing and the second by the yellow tracing. In this case the stimulation pulse has the same parameters of the previous demonstration, namely a primary amplitude of 700 mV and the remaining parameters left as preset.

The reason why Figures 6.9 and 6.10 have a different look than the other results is because they were obtained with a different oscilloscope.

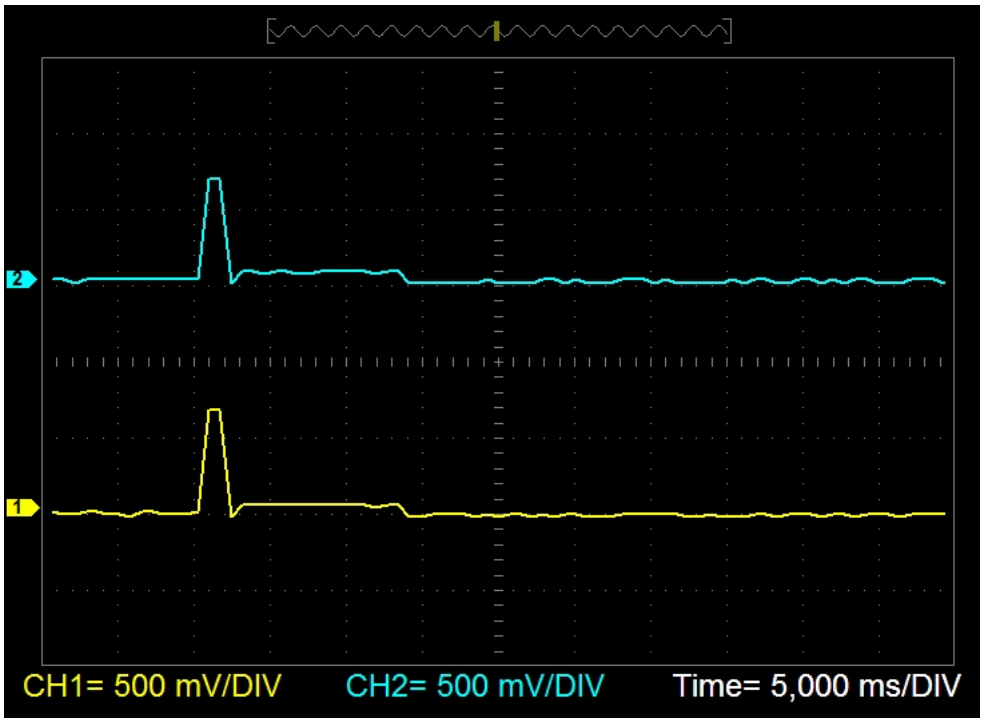

FIGURE 6.9: Synchronous rectangular-based stimulation.

From the results shown in Figure 6.7, a measure of the duration of the phases was also made to check that the configured timings were being met. The measured timings were the widths of the charge and charge-balancing phases from the three pulses, as can be seen in Figure 6.11. The results obtained are shown in Table 6.2. As the values for the charge width was left as preset, i.e., a width of 1 ms, with an also preset biphasic ratio of 10, the expected charge-balancing was of 10 ms. However, as it is possible to see in Table 6.2, these widths were not met in the first pulse but were approximately correct in the following ones. Besides this, it is possible to see that the widths show a trend to increase along the repetitions. The cause for these differences is related to the clock frequency of the microcontroller and also the delay caused by some functions of the microcontroller.

6.2 Charge Balancing Analysis

The charge-balancing capacity of this system was also tested by applying a simple stimulation profile to a dummy load composed of an 8.2 kOhm resistor and an 8.8 μF capacitor in series and measuring the voltage across the capacitor. Ideally, at the end of the application of stimulation, the result of this test should be that the voltage potential on one

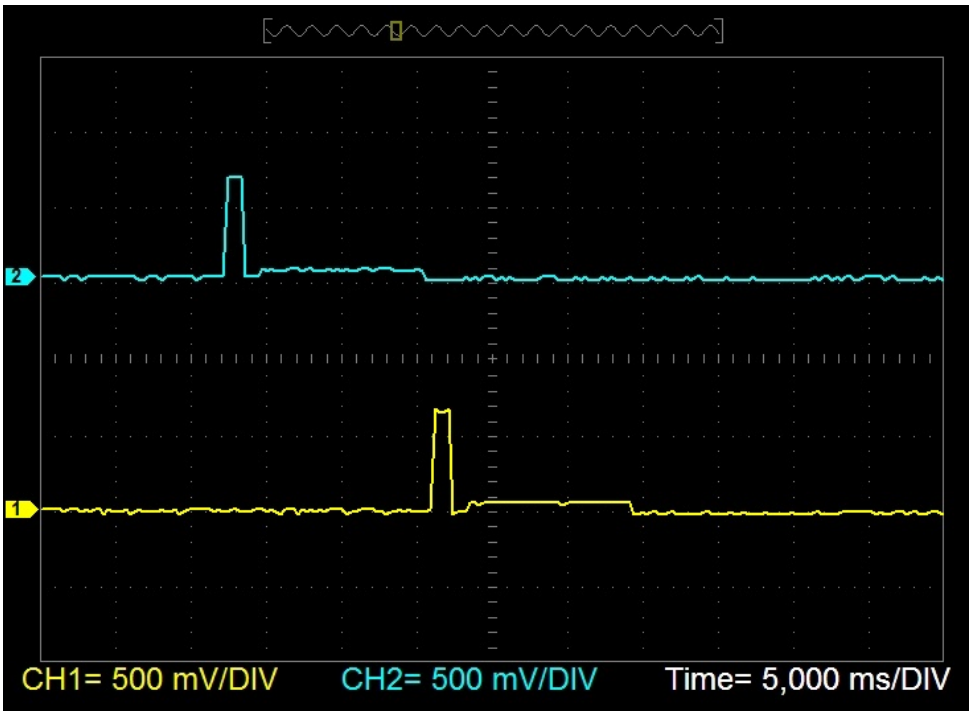

FIGURE 6.10: Asynchronous rectangular-based stimulation.

TABLE 6.2: Timing measurements. The definition of the intervals is shown previously in Figure 6.11.

Interval	**Width [ms]**
A	0.780
B	8.08
C	1.08
D	10.50
E	1.08
F	10.72

side of the capacitor would be the same as on the other side, that is, there is no potential difference across it meaning that there is no charge accumulated in the capacitor. For this test, four probes were used: a green one for the switch input corresponding to cathodic conformation, a yellow one for the switch input corresponding to anodic conformation, a blue one for the cathode side of the capacitor and a pink one for the anode side of the capacitor. It is important to mention that the green probe was multiplying the signal by 10 and this option could not be switched off. Also it is important to refer that some charge can escape through the probes. Figures 6.12, 6.13, 6.14, 6.15 and 6.16 show the results obtained for the five preset waveforms.

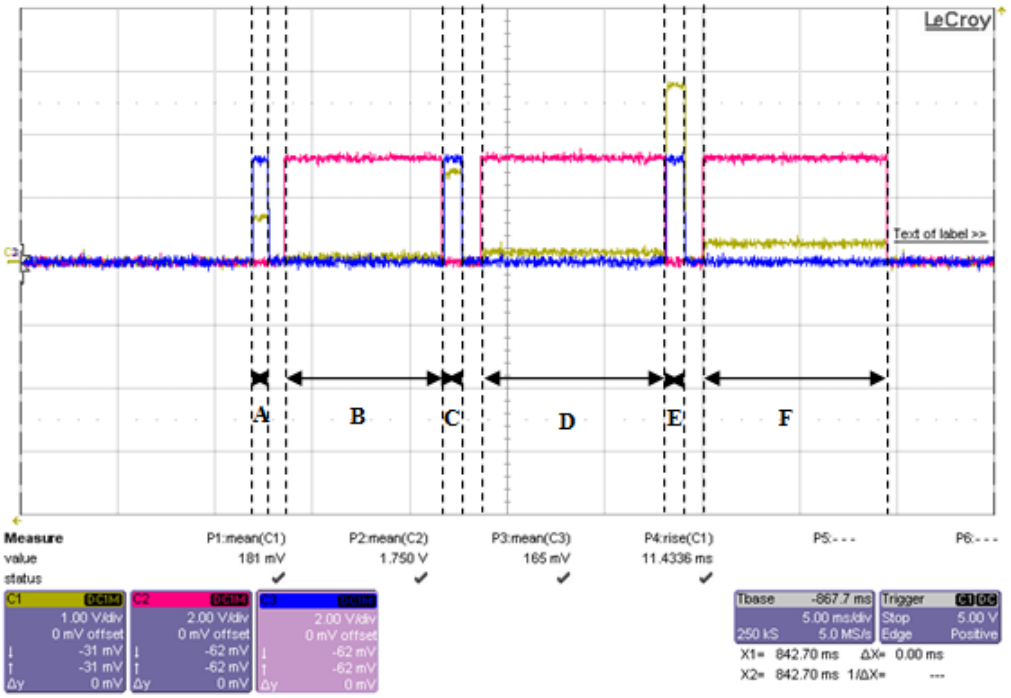

FIGURE 6.11: Intervals for timing measurement.

In Figure 6.12 the results from a single rectangular shaped stimulation are presented. As it is possible to see, at the end of the stimulation, which corresponds to the end of the yellow line, the blue and pink probes measure approximately the same voltage level, meaning that charge-balancing across the capacitor is correctly achieved.

FIGURE 6.12: Charge-balancing results for Rectangular shaped stimulation.

In Figure 6.13 the results from a single bi-level Lilly shaped stimulation are presented. As it is possible to see in this case, at the end of the stimulation, the blue and pink probes measure a slightly different voltage, meaning that some charge build-up occurs on the capacitor and thus a correct charge-balancing is not completely achieved.

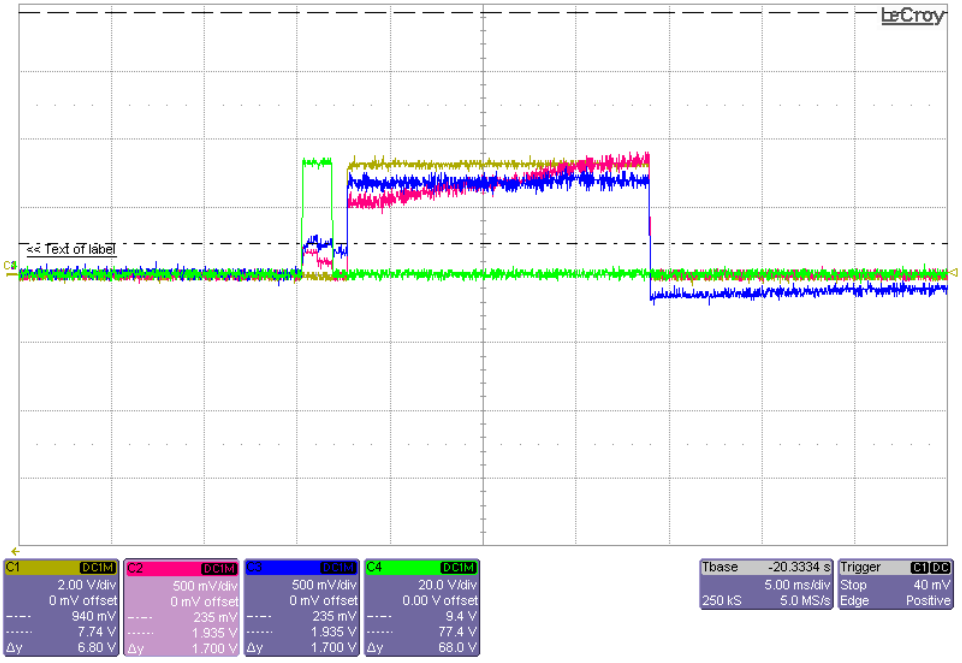

FIGURE 6.13: Charge-balancing results for Bi-level Lilly shaped stimulation.

In Figure 6.14 the results from a single quasi-trapezoidal shaped stimulation are presented. As it is possible to see in this case, at the end of the stimulation, the blue and pink probes measure a slightly different voltage, meaning that some charge build-up occurs on the capacitor and thus a correct charge-balancing is not completely achieved.

In Figure 6.15 the results from a single tri-exponential shaped stimulation are presented. As it is possible to see, at the end of the stimulation, the blue and pink probes measure approximately the same voltage level, meaning that charge-balancing across the capacitor is correctly achieved.

In Figure 6.16 the results from single bursting strategy based stimulation are presented. As it is possible to see, at the end of all stimulation, blue and pink probes measure approximately the same voltage level, meaning that charge-balancing across the capacitor is correctly achieved.

The results from this charge-balancing test are presented in Table 6.3. In this table are the approximate amplitude values measured at the end of the stimulation for both

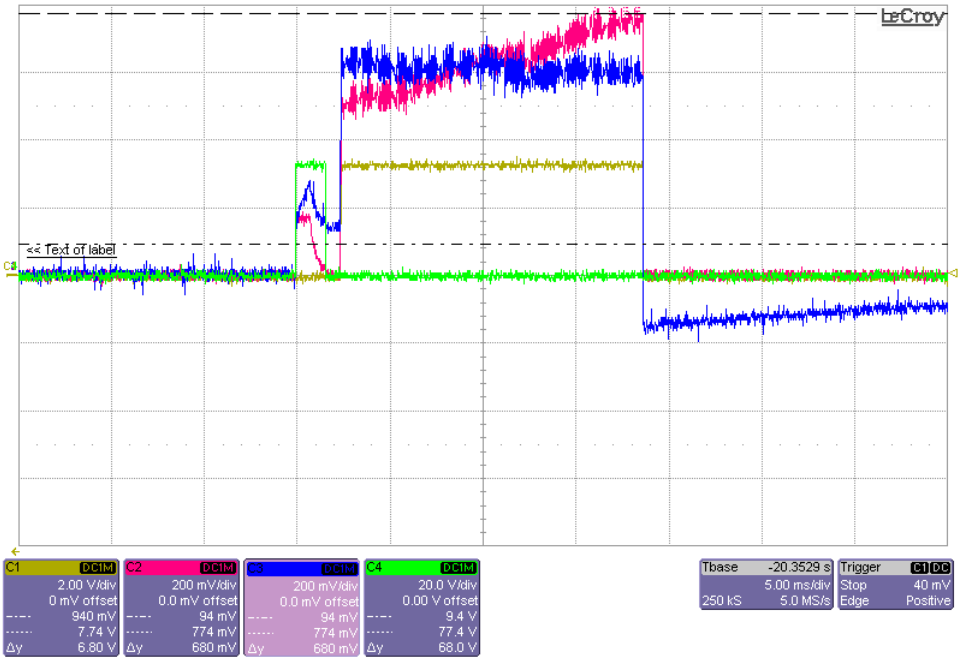

FIGURE 6.14: Charge-balancing results for Quasi-Trapezoidal based stimulation.

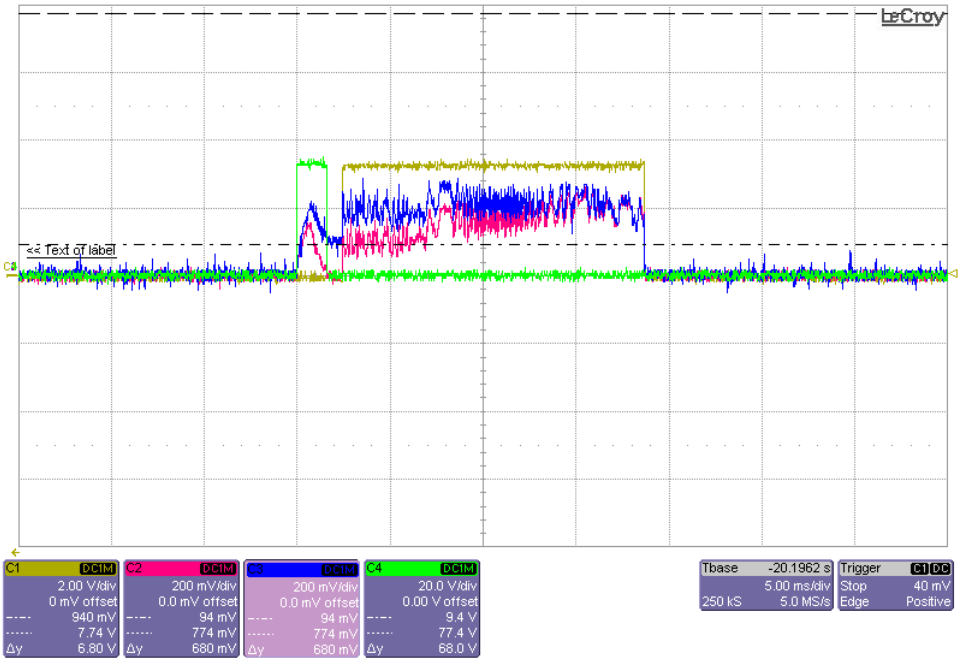

FIGURE 6.15: Charge-balancing results for Tri-Exponential based stimulation.

cathodic and anodic measuring sites, the difference between them and the corresponding residual charge in the capacitor.

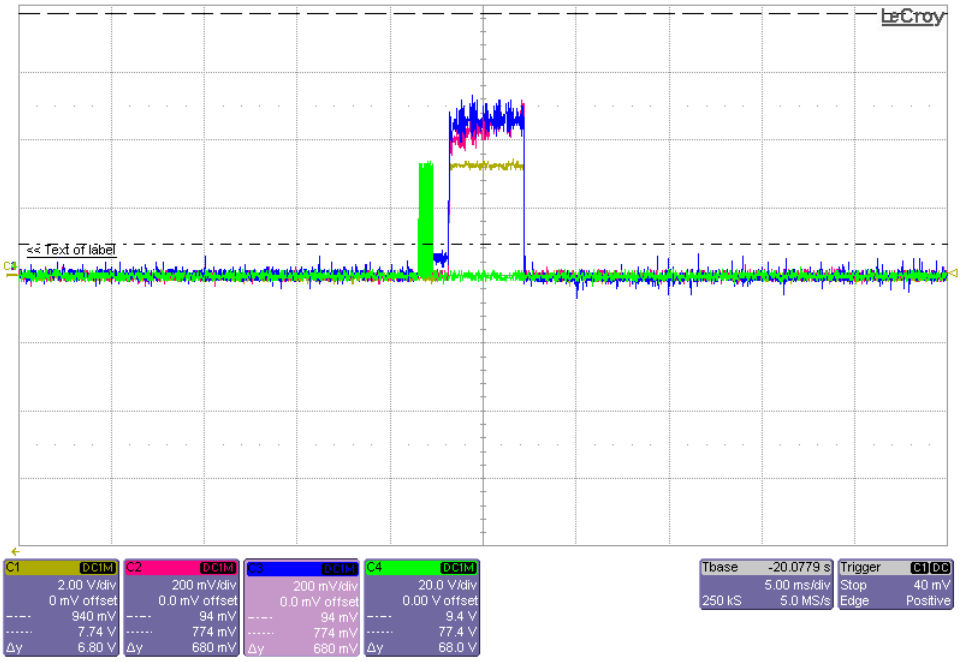

FIGURE 6.16: Charge-balancing results for Bursting Strategies based stimulation.

TABLE 6.3: Charge Balance measurements.

Waveform Type	**Cathodic Value [mV]**	**Anodic Value [mV]**	**Cathodic-Anodic**	**Residue Charge [μC]**
Rectangular	500	500	0	0
Bi-level Lilly	700	800	- 100	- 0.9
Quasi-Trapezoidal	600	774	- 100	- 1
Tri-Exponential	200	200	0	0
Bursting Strategy	450	450	0	0

These results show that this system can provide a correctly charge-balanced stimulation in three of the five stimulation based on preset waveforms, namely with Rectangular waveform, Tri-Exponential and Bursting Strategies. In Bi-level Lilly and Quasi-Trapezoidal however, there is a difference between the voltages on both sides of the capacitor, meaning that there is some charge being accumulated there, more precisely, a charge of -0.9 μC for Bi-level Lilly waveform and -1 μC for Quasi-Trapezoidal. Despite the existence of this residual charge, it is not considered big enough to dismiss the charge-balancing capacity for both waveforms. The reason for not dismissing its capacity is the fact that safety range, known as water window, for example for a platinum electrode and a Ag/AgCl reference electrode is from -0.6 V to 0.8 V, and when a $Hg/HgCl_2$ reference electrode, also known calomel electrode, is used the range goes

from -0.8 V up to 1.2 V [11]. Beyond the limits of the water window, hydrolysis and electrode dissolution can occur. Since the voltage difference for Bi-level Lilly and Quasi-Trapezoidal waveforms is of -100 mV and –144 mV, respectively, it means that it would be within these safety limits. Besides this, the fact that repetition of stimulation was found to lead to increasingly smaller charge difference [37] gives reasons to believe that this result can be improved.

Chapter 7

Conclusion and Future Work

As an area of major interest due to its various possible applications, from therapy to treatment, to rehabilitation and also prosthetics, neurostimulation research has lead to the development of many currently used technologies in the aforementioned areas. Some of the resulting technologies are nowadays well-known and widely used such as the pacemaker for cardiac control and the cochlear implant for people with hearing loss. Besides these two technologies others less known are also already being used for therapies, treatments and rehabilitation and in functional prostheses and more are on the way.

For the development of these technologies in its mentioned applications, careful studies of stimulation strategies have to be done so that an effective, safe and energy-efficient stimulation can be obtained for each specific use. For an optimal stimulation to be obtained, comparative studies have to be done between different stimulation strategies which ideally should be within a relatively small period of time, so that the nerve's conditions do not change, and with the minimum changes to the experimental apparatus, so that the conditions of the experience stay the same. However, the classical methods for neural stimulation studies did not enable these requirements to be met since it took time and amendments to the apparatus to change the parameters of the stimulation. To meet the comparative studies requirements, new stimulation experimental apparatus have been developed firstly with testing platforms with multiple customized circuits for stimulations with different amplitude and duration for a specific waveform [18] and computational models [24], and then more advanced systems that allowed for a increasingly

easier and faster programmability of stimulation with an higher number of parameters available for configuration [25-34, 36].

The project reported in this thesis is an approach to create an improved, small, low-cost, effective, efficient and easy-to-use version of a programmable stimulation system for these comparative studies of stimulation strategies. In addition to this it is also a purpose that this project can be easily recreated, changed and, eventually, improved.

Throughout this thesis we have described the development of an embedded platform for electrical neural stimulation that allows its user to control the parameters of stimulation through a MATLAB-based user-system interface, as explained in Chapter 4.1, and is then capable of generating and applying a charge-balancing bipolar stimulation via four different channels thanks to a microcontroller and a front-end circuit board, as explained in Chapter 4.2 and 5. The results reported in Chapter 6 show that the platform here developed can operate correctly in both the main aspects of its functioning: the capacity to generate the stimulation according to all the parameters as configured by the user and the capacity to provide a charge-balancing stimulation in all the preset waveforms.

When compared to the existing programmable stimulation platforms, the system here developed presents several advantages. Besides being a small, low-cost, easy-to-build, effective and easy-to-use stimulation system, the platform developed for this thesis also presents to its users a higher level of programmability. Regarding the configuration of stimulation characteristics, this system allows users to choose between five preset waveforms or upload a arbitrary waveform, to control pulse parameters such as amplitude, width, decay width, exponential level of decay and time resolution, to control the delay between charge phase and charge-balancing phase – interphasic delay, the ratio between charge amplitude and charge-balancing amplitude – biphasic ratio, to define the repetition ratio and the possibility of the pulse amplitude vary in each repetition – repetition amplitude ratio, and the duration intended for the full stimulation. In addition to this, the platform here developed allows the user to apply stimulation in three different ways with the possible use of the four available stimulation channels, namely using only one channel, using more than one sequentially or more than one at the same time.

However, there are still some significant limitations to the platform here developed. One significant limitation is related to the maximum output that the system can provide. The first output limitation comes from the DAC which can only output a maximum of

5 V which means that users cannot configure in this system stimulations with voltage amplitude higher than 5 V. The second output limitation comes from the opamp voltage compliance, which is of ±9 V – the power supply values. This is a limiting factor because it means that the electrode impedance has major role on the maximum voltage that the user can configure for the stimulation since that beyond the voltage compliance point the opamp saturates and the configurations defined are not met. However, this limitation may not be so significant because this platform was originally thought for experimental setups where the electrodes are placed closer to the target nerve where stimulation usually has lower amplitudes. Table 7.1 presents all the ranges and specifications of the platform developed.

Table 7.1: Specifications of the developed platform.

Parameter	**Value**	**Units**
Amplitude Maximum	5	[V]
Amplitude Resolution	0.01	[V]
Charge Phase Width	0.1 - 11.6	[ms]
Interphasic Delay	0 – 11.6	[ms]
Time Resolution	0.025 – 1	[ms]
Biphasic Ratio	0 – 221	-
Repetition Frequency	0.001 – 40	[kHz]

A second limitation is related to its failure to comply exactly with the pulse widths and the fact that these widths increase along the repetitions. These width problems are thought to be caused by the clock frequency of the microcontroller and also the delay caused by some functions of the microcontroller, meaning that an optimization of the microcontroller's code together with its clock frequency is necessary.

A third limitation of this system is the energy consumption it presents. During the tests performed the operation of the front-end board alone, including when a stimulation was done, the current consume was of about 300 mA, with peaks reaching 500 mA. Although this is not problematic for the experimental procedure itself, it means a high energy consume from the power source and consequently a shorter lifetime.

Another limitation of the system here developed is the fact that all connections, between computer, microcontroller and the front-end board, is done through wires. This presents

as a limiting factor because it reduces the liberty of movements and maneuver that can be done in an experimental setup using this platform.

Table 7.2 presents a comparison between the embedded platform described in this thesis and the five most recent stimulation systems mentioned in Chapter 3. For this table focus was made on the following characteristics: number of waveforms available for user choice; the possibility of creating a custom waveform, different from the available ones; the number of parameters available for configuration; if the system allows for biphasic stimulation; the number of channels available to apply stimulation; the number of modes for applying stimulation – firing modes; the electric mode of applying stimulation; if the system can work in stand-alone mode, i.e., if it does not need for a computer controlling its operation; and if the communication can be made wirelessly.

TABLE 7.2: Comparison between this work and existing systems.

Characteristics	**This Work**	**[30]**	**[31]**	**[33]**	**[34]**	**[36]**
No Waveforms	5	1	5	1	1	2
Custom Waveform	Yes	No	Yes	No	No	Yes
No Parameters	20	4	5	4	6	16
Biphasic	Yes	Yes	Yes	Yes	No	Yes
No Channels	4	100	1	8	2	4
Firing Modes	3	2	3	1	1	2
Electric Mode	Current -Controlled	Current -Controlled	Current -Controlled	Voltage or Current	Voltage -Controlled	Voltage -Controlled
Stand-Alone Mode	No	No	Yes	No	No	Yes
Wireless	No	Yes	No	No	Yes	No

The challenges for the future of this platform are then the search for the solutions for these limitations: to improve its output capacity both by a different, more powerful DAC and an improved V-to-I converter circuit that can overcome these voltage compliance limits, to optimize the microcontroller code together with its clock frequency so that stimulations width can be exactly as intended by the user, and finally to make it less power consuming and make the communication between computer and microcontroller wireless so that microcontroller and front-end board can be used for example as an implantable stimulating platform.

References

[1] Famm, K., et al. (2013). "Drug discovery: a jump-start for electroceuticals." Nature 496(7444): 159-161.

[2] Ragnarsson, K. T. (2008). "Functional electrical stimulation after spinal cord injury: current use, therapeutic effects and future directions." Spinal Cord 46(4): 255-274.

[3] Yawn, R., et al. (2015). "Cochlear implantation: a biomechanical prosthesis for hearing loss." F1000Prime Rep 7: 45.

[4] "A diagram of Chalmers' implanted prosthetics system" from http://www.gizmag.com/thought-controlled-prosthetic-arm/25216/ in 07/12/2014

[5] Seely R., Stephens T., Tate P. (2011). "Anatomia e Fisiologia" (8th ed.). Loures: Lusociência.

[6] "Neuron diagram" from https://www.pinterest.com/beverleycain9/neuroscience/ in 12/06/2015

[7] Plonsey, R. and R.C. Barr (2007). Bioelectricity, a quantitative approach., Springer, 3rd ed

[8] "Diagram of an action potential" from http://www.cog.brown.edu/courses/cg0001/lectures/visualpaths.html in 29/11/2014

[9] "Variation of Membrane Potential during an Action Potential" from http://hmphysiology.blogspot.pt/p/nervous-system.html in 02/09/2015

[10] Luan, S., et al. (2014). "Neuromodulation: present and emerging methods." Front Neuroeng 7: 27.

[11] Liu, X., A. Demosthenous, N. Donaldson. (2011). "Neural Interfaces for Implanted Stimulators." In: Springer Handbook of Medical Technology., Springer: 749-766.

[12] Merrill, D. R., et al. (2005). "Electrical stimulation of excitable tissue: design of efficacious and safe protocols." J Neurosci Methods 141(2): 171-198.

[13] Dymond, A. M. (1976). "Characteristics of the metal-tissue interface of stimulation electrodes." IEEE Trans Biomed Eng 23(4): 274-280.

[14] de Voogt, W. G., et al. (2004). "Understanding capture detection." Europace 6(6): 561-569.

[15] Prado-Guitierrez, P., et al. (2006). "Effect of interphase gap and pulse duration on electrically evoked potentials is correlated with auditory nerve survival." Hear Res 215(1-2): 47-55.

[16] W. M. Grill and J. T. Mortimer (1995), "Stimulus waveforms for selective neural stimulation," IEEE Eng. Med. Biol. Mag., 14(4): 375–385.

[17] Fang, Z. P. and J. T. Mortimer (1991). "Selective activation of small motor axons by quasi-trapezoidal current pulses." IEEE Trans Biomed Eng 38(2): 168-174.

[18] Accornero, N., et al. (1977). "Selective Activation of peripheral nerve fibre groups of different diameter by triangular shaped stimulus pulses." J Physiol 273(3): 539-560.

[19] van den Honert, C. and J. T. Mortimer (1979). "Generation of unidirectionally propagated action potentials in a peripheral nerve by brief stimuli." Science 206(4424): 1311-1312.

[20] Grill, W. M. and J. T. Mortimer (1997). "Inversion of the current-distance relationship by transient depolarization." IEEE Trans Biomed Eng 44(1): 1-9.

[21] Qing, K., et al. (2015). "Burst-modulated waveforms optimize electrical stimuli for charge efficiency and fiber selectivity." IEEE Trans Neural Syst Rehabil Eng.

[22] Qing, K, (2013). "Electrical stimulation with burst-modulated pulse waveforms improves efficiency and fiber-selectivity." In 6th International IEEE EMBS Conference on Neural Engineering. San Diego, CA, November 6-8 2013. USA: IEEE. 1.

[23] Kilgore, K. L. and N. Bhadra (2004). "Nerve conduction block utilising high-frequency alternating current." Med Biol Eng Comput 42(3): 394-406.

[24] Grill, W. M., Jr. and J. T. Mortimer (1996). "The effect of stimulus pulse duration on selectivity of neural stimulation." IEEE Trans Biomed Eng 43(2): 161-166.

[25] Bugbee, M., et al. (2001). "An implant for chronic selective stimulation of nerves." Med Eng Phys 23(1): 29-36.

[26] Sha, H., et al. (2005). "A microcontroller-based implantable nerve stimulator used for rats." Conf Proc IEEE Eng Med Biol Soc 6: 6176-6179.

[27] Salmons, S., et al. (2001). "ASIC or PIC? Implantable stimulators based on semi-custom CMOS technology or low-power microcontroller architecture." Med Eng Phys 23(1): 37-43.

[28] Constandinou, T. G., et al. (2008). "A partial-current-steering biphasic stimulation driver for vestibular prostheses." IEEE Trans Biomed Circuits Syst 2(2): 106-113.

[29] Woods, V., et al. (2008). "A Reconfigurable and Automated System for the study of Stimulus Waveform Parameters." Conf Proc IFESS 2008

[30] Thurgood, B. K., et al. (2009). "A wireless integrated circuit for 100-channel charge-balanced neural stimulation." IEEE Trans Biomed Circuits Syst 3(6): 405-414.

[31] Eftekhar, A., et al. (2009). "A Programmable Neural Interface for Investigating Arbitrary Stimulation Strategies" Conf Proc IFESS 2009

[32] Guilvard, A., et al. (2010) "A Fully-Programmable Neural Interface for Multi-Polar, Multi-Channel Stimulation Strategies." IEEE Circuits Sys 2010: 2235-2238.

[33] Laotaveerungrueng, N., et al. (2010). "A high-voltage, high-current CMOS pulse generator ASIC for deep brain stimulation." Conf Proc IEEE Eng Med Biol Soc 2010: 1519-1522.

[34] Chiu, H.-C., et al. (2014). "An Intelligent Brain Machine Interface with Wireless Micro-Stimulation and Neural Recording." IEEE Biomed Health Inform 2014.

[35] http://www.open-ephys.org/ in 23/09/2015

[36] https://sites.google.com/site/pulsepalwiki/home in 23/09/2015

[37] Williams, I. and T. Constandinou (2013). "An energy-efficient, dynamic voltage scaling neural stimulator for a proprioceptive prosthesis." IEEE Trans Biomed Circuits Syst 7(2): 129-139.

Appendix A

MATLAB Code

A.1 User-System Interface Code

```
function varargout = stimulation_gui(varargin)
% STIMULATION_GUI MATLAB code for stimulation_gui.fig
%      STIMULATION_GUI, by itself, creates a new STIMULATION_GUI or raises the existing
%      singleton*.
%
%      H = STIMULATION_GUI returns the handle to a new STIMULATION_GUI or the handle to
%      the existing singleton*.
%
%      STIMULATION_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in STIMULATION_GUI.M with the given input arguments.
%
%      STIMULATION_GUI('Property','Value',...) creates a new STIMULATION_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before stimulation_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to stimulation_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% edit the above text to modify the response to help stimulation_gui

```

```
25 % Last Modified by GUIDE v2.5 17-Mar-2015 11:56:17
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30     'gui_Singleton',  gui_Singleton, ...
31     'gui_OpeningFcn', @stimulation_gui_OpeningFcn, ...
32     'gui_OutputFcn',  @stimulation_gui_OutputFcn, ...
33     'gui_LayoutFcn',  [] , ...
34     'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 % --- Executes just before stimulation_gui is made visible.
48 function stimulation_gui_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to stimulation_gui (see VARARGIN)
54
55 % Initial Considerations
56
57 handles.t_i = 0; %1; % just to see the initial fase of the stimulus
58 handles.t_f = 0;    %2; % just to see the end of the stimulus
59 handles.amplitude_1 = 1;
60 handles.amplitude_2 = 0;
61 handles.duration_1 = 1; % 1ms (100/1000) stimulation
62 handles.duration_2 = 0;
63 handles.amplitude_I0 = 0.01; % for tri-exponential growth, since I0 cannot be
       zero is a value aprox. zero
64 handles.delay = 1;
65 handles.decay = 1;
66 handles.biphasic = 10;
67 handles.resolution = 0.25;
68 handles.time_ms_s = 1/1000;
69 handles.time_s_ms = 1000;
70 handles.time_ms_us = 1000;
71 handles.q = 1;
```

```
72  handles.q2 = 2.197e+03;
73  handles.a = 0.2;
74  handles.a2 = 1.197e+03;
75  handles.b = 0.3;
76  handles.b2 = 2.197e+03;
77  handles.c = 0.5;
78  handles.c2 = 3.197e+03;
79  handles.decay_steps = handles.decay/handles.resolution;
80  handles.burst_steps = handles.duration_1/handles.resolution;
81  handles.increase_steps = handles.duration_1/handles.resolution;
82  handles.repetition = 0; % starts with no repetition, just the original wave
83  handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
84  handles.duration_balance_2 = handles.biphasic.*handles.duration_2;
85  handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
        duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
86  handles.time_resolution = 40; % so 0,025ms * 40 = 1 time sample
87  handles.timescale = (0:(1/handles.time_resolution):(handles.stimulation_fulltime+
        handles.t_f));
88  handles.current_data_I = zeros(1,length(handles.timescale));
89  handles.stimulation_data_I = zeros(1,length(handles.timescale));
90  handles.fullstimulation_duration = handles.stimulation_fulltime;
91  handles.rep_amp_ratio = 1;
92  handles.time_lag = 1;
93  handles.number_sent_channels = 1;
94  handles.old_channel = 0;
95  handles.interstimulus_time = 0;
96  handles.n_repetitions = 0;
97  handles.old_firing = 0;
98  handles.amplitude_max = 0;
99  handles.reps_max = 0;
100
101 handles.resistance = 0;
102 handles.DAC_amplitude_limit = 5; % CHANGE HERE IF HIGHER LIMIT IS DESIRED
103 handles.opamp_max_output_voltage = 9;
104 handles.R_set = 510; % it was initially tought with 500 but the actual resistor
        is 510
105 handles.R_internal = 12;
106
107
108
109 set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        fullstimulation_duration)
110
111 serialInfo = instrhwinfo('serial');
112 if isempty(serialInfo.SerialPorts) == 1
113     % Nothing happens and program moves on
114 else
115     % if finds COM ports being used, reports them and makes them available
```

```
116     % for user selection
117     n_coms = 1;
118     for n_coms = 1:length(serialInfo.SerialPorts)
119         string_list(n_coms+1) = serialInfo.SerialPorts(n_coms);
120     end
121     set(handles.COM_popup,'String',string_list);
122 end
123
124 % User notification of Type of System
125 input_answer = inputdlg('This system is an Open Loop Control. Enter Electrode
        Impedance Value (Ohm) at 1kHz for Amplitude Safe Range: [Cancel for No Safe
        Range]', 'Safe Range Amplitude');
126 if isempty(input_answer) == 1
127     % If the User chooses not to have a Safe Range
128     wd = warndlg('No Amplitude Safe Range Defined!','Safe Range');
129     waitfor(wd);
130 else
131     % Calculates the Amplitude Maximum for a Safety
132     try
133         handles.resistance = str2num(cell2mat(input_answer));
134         handles.amplitude_max = (handles.opamp_max_output_voltage * handles.R_set
        )/(handles.resistance + handles.R_set + (2*handles.R_internal));
135         set(handles.electrode_imped_edit, 'enable', 'on','String', handles.
        resistance)
136         set(handles.amplitude_maximum_edit, 'enable', 'off','String', handles.
        amplitude_max)
137     catch err
138         wd = warndlg('No Amplitude Safe Range Defined!','Safe Range');
139         waitfor(wd);
140         handles.resistance = 0;
141         handles.amplitude_max = 0;
142     end
143 end
144
145
146
147
148 % Waveform signals
149
150 % Pulse Waveform
151 handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-handles.
        amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
152     zeros(1,(handles.delay*handles.time_resolution))...
153     ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones(1,handles.
        duration_balance_1*handles.time_resolution)...
154     zeros(1,(length(handles.timescale)-handles.t_i*handles.time_resolution)...
155     -((handles.stimulation_fulltime)*handles.time_resolution))];
156
```

```
157
158 % Bi-Level Lilly Waveform
159 % amplitude_CB is the amplitude needed for charge balance
160 amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.amplitude_2*
        handles.duration_2)/handles.biphasic;
161 handles.duration_balance_total = handles.biphasic.*(handles.duration_1 + handles.
        duration_2);
162 handles.stimulation_fulltime = handles.duration_1+handles.duration_balance_total+
        handles.delay+handles.decay;
163 handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-handles.
        amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
164     (-handles.amplitude_2).*ones(1,handles.duration_2*handles.time_resolution)
        zeros(1,(handles.delay*handles.time_resolution))...
165     amplitude_CB.*ones(1,handles.duration_balance_total*handles.time_resolution)
        ...
166     zeros(1,(length(handles.timescale)-handles.t_i*handles.time_resolution)...
167     -((handles.stimulation_fulltime)*handles.time_resolution))];
168
169
170
171 % Quasi-Trapezoidal Waveform
172 handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-handles.
        amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
173 i=1;
174 total_decay_amplitude = 0;
175 %Building of the decay part of wave
176 for i=1:handles.decay_steps
177     handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.amplitude_1*handles
        .q*...
178         exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s)).*...
179         ones(1,handles.resolution*handles.time_resolution));
180     total_decay_amplitude = total_decay_amplitude + (handles.amplitude_1*handles.
        q*...
181         exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
182 end
183 %Building of Discharge Phase
184 amplitude_CB = (handles.amplitude_1*handles.duration_1 + total_decay_amplitude*
        handles.resolution)/handles.biphasic;
185 handles.duration_balance_total = handles.biphasic.*(handles.duration_1 + handles.
        decay);
186 handles.stimulation_fulltime = handles.duration_1+handles.duration_balance_total+
        handles.delay+handles.decay;
187 handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.delay*handles.
        time_resolution))...
188     amplitude_CB.*ones(1,handles.duration_balance_total*handles.time_resolution)
        ...
189     zeros(1,(length(handles.timescale)-handles.t_i*handles.time_resolution)...
190     -((handles.stimulation_fulltime)*handles.time_resolution))]);
```

```
191
192
193 % Tri-Exponetial Waveform
194
195 handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
196 j_i = 1;
197 j_d = 1;
198 total_increase_amplitude = 0;
199 total_decay_amplitude = 0;
200 %Building of the increase part of wave
201 for j_i=1:handles.increase_steps
202     handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.amplitude_1)/((
      handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.resolution*handles
      .time_resolution)...
203         +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+handles.t_i)-
      handles.t_i)))/handles.time_resolution).*...
204         ones(1,handles.resolution*handles.time_resolution));
205     total_increase_amplitude = total_increase_amplitude + (((handles.amplitude_1)
      /((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.resolution*
      handles.time_resolution)...
206         -((handles.amplitude_1*handles.t_i)/(handles.duration_1+handles.t_i)-
      handles.t_i))/handles.time_resolution);
207
208 end
209 %Building of the decay part of wave
210 for j_d=1:handles.decay_steps
211     handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((handles.t_i+
      handles.duration_1)*handles.time_resolution)*((handles.a*...
212         exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))+(handles.b
      *...
213         exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))+(handles.c
      *...
214         exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))).*...
215         ones(1,handles.resolution*handles.time_resolution));
216     total_decay_amplitude = total_decay_amplitude + (handles.tri_exp((handles.t_i
      +handles.duration_1)*handles.time_resolution)*((handles.a*...
217         exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))+(handles.b
      *...
218         exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))+(handles.c
      *...
219         exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s))));
220 end
221 %Building of Discharge Phase
222 amplitude_CB = (total_increase_amplitude*handles.resolution +
      total_decay_amplitude*handles.resolution)/handles.biphasic;
223 handles.duration_balance_total = handles.biphasic.*(handles.duration_1 + handles.
      decay);
```

```
224 handles.stimulation_fulltime = handles.duration_1+handles.duration_balance_total+
        handles.delay+handles.decay;
225 handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*handles.
        time_resolution))...
226     amplitude_CB.*ones(1,handles.duration_balance_total*handles.time_resolution)
        ...
227     zeros(1,(length(handles.timescale)-handles.t_i*handles.time_resolution)...
228     -((handles.stimulation_fulltime)*handles.time_resolution))]);
229
230
231 % Bursting Waveform
232 handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
233 k=1;
234 %Building of the steps
235 for k=1:handles.burst_steps
236     handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1.*...
237         ones(1,(handles.resolution/2)*handles.time_resolution) ...
238         zeros(1,(handles.resolution/2)*handles.time_resolution)]);
239 end
240 %Building of Discharge Phase
241 handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*handles.
        time_resolution))...
242     ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones(1,handles.
        duration_balance_1*handles.time_resolution)...
243     zeros(1,(length(handles.timescale)-handles.t_i*handles.time_resolution)...
244     -((handles.stimulation_fulltime)*handles.time_resolution))]);
245
246
247 % Initial Plot Output Look
248 axes(handles.plot_axes);
249 plot(handles.timescale,handles.current_data_I);
250 ylim([(min(handles.current_data_I)-0.1),-(min(handles.current_data_I)-0.1)]);
251 xlim([0,((length(handles.current_data_I)-1)/handles.time_resolution)]);
252 ylabel('Amplitude (V)');
253 xlabel('Time (ms)');
254
255 % Choose default command line output for stimulation_gui
256 handles.output = hObject;
257
258 % Update handles structure
259 guidata(hObject, handles);
260
261 % UIWAIT makes stimulation_gui wait for user response (see UIRESUME)
262 % uiwait(handles.figure1);
263
264
265 % --- Outputs from this function are returned to the command line.
266 function varargout = stimulation_gui_OutputFcn(hObject, eventdata, handles)
```

```
267 % varargout  cell array for returning output args (see VARARGOUT);
268 % hObject    handle to figure
269 % eventdata  reserved - to be defined in a future version of MATLAB
270 % handles    structure with handles and user data (see GUIDATA)
271
272 % Get default command line output from handles structure
273 varargout{1} = handles.output;
274
275
276 % --- Executes on selection change in plot_popup.
277 function plot_popup_Callback(hObject, eventdata, handles)
278 % hObject    handle to plot_popup (see GCBO)
279 % eventdata  reserved - to be defined in a future version of MATLAB
280 % handles    structure with handles and user data (see GUIDATA)
281
282 % Hints: contents = cellstr(get(hObject,'String')) returns plot_popup contents as
        cell array
283 %        contents{get(hObject,'Value')} returns selected item from plot_popup
284
285 % DO NOT CHANGE THIS POPUP PART IN .FIG AGAIN
286
287 val = get(handles.plot_popup, 'Value');
288 handles.plot_popup_value = val;
289 str = get(handles.plot_popup, 'String');
290 handles.plot_popup_string = str;
291 switch str{val}
292     case 'Rectangular' % User selects pulse wave
293
294         % Defines variables to the original value
295         handles.amplitude_1 = 1;
296         handles.duration_1 = 1;
297         handles.amplitude_2 = 0;
298         handles.duration_2 = 0;
299         handles.biphasic = 10;
300         handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
301         handles.duration_balance_2 = 0;
302         handles.decay = 0;
303         handles.delay = 1;
304         handles.resolution = 0.25;
305         handles.repetition = 0;
306         handles.rep_amp_ratio = 1;
307         handles.stimulation_fulltime = handles.duration_1+handles.duration_2+
      handles.duration_balance_1+handles.duration_balance_2+handles.delay+handles.
      decay;
308         handles.fullstimulation_duration = handles.stimulation_fulltime;
309         handles.timescale = (0:(1/handles.time_resolution):(handles.
      stimulation_fulltime+handles.t_f));
310
```

```
311         %Building the pulse wave
312         handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-handles.
    amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
313             zeros(1,(handles.delay*handles.time_resolution))...
314             ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones(1,
    handles.duration_balance_1*handles.time_resolution)...
315             zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
316             -((handles.stimulation_fulltime)*handles.time_resolution))];
317         %
318         handles.current_data_I = handles.pulse;
319         handles.stimulation_data_I = handles.pulse;
320
321         % Define which edit fields should work and define its initial values
322         set(handles.amplitude1_edit, 'enable', 'on','String', handles.amplitude_1
    )
323         set(handles.duration1_edit, 'enable', 'on','String', handles.duration_1)
324         set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
325         set(handles.delay_edit, 'enable', 'on','String', handles.delay)
326         set(handles.amplitude2_edit, 'enable', 'off')
327         set(handles.duration2_edit, 'enable', 'off')
328         set(handles.decay_edit, 'enable', 'off')
329         set(handles.resolution_edit, 'enable', 'off')
330         set(handles.repetition_edit, 'enable', 'on','String', handles.repetition)
331         set(handles.Q_exp_edit, 'enable', 'off')
332         set(handles.Q2_exp_edit, 'enable', 'off')
333         set(handles.A_exp_edit, 'enable', 'off')
334         set(handles.A2_exp_edit, 'enable', 'off')
335         set(handles.B_exp_edit, 'enable', 'off')
336         set(handles.B2_exp_edit, 'enable', 'off')
337         set(handles.C_exp_edit, 'enable', 'off')
338         set(handles.C2_exp_edit, 'enable', 'off')
339         set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
340         set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
341
342     case 'Bi-level Lilly' % User selects bilevel Lilly
343         handles.amplitude_1 = 1;
344         handles.duration_1 = 1;
345         handles.amplitude_2 = 0.5;
346         handles.duration_2 = 1;
347         handles.biphasic = 10;
348         handles.delay = 1;
349         handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
350         handles.duration_balance_2 = handles.biphasic.*handles.duration_2;
351         handles.decay = 0;
352         handles.resolution = 0.25;
```

```
353         handles.repetition = 0;
354         handles.rep_amp_ratio = 1;
355 
356         % Building the Bilevel Lilly
357         amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
        amplitude_2*handles.duration_2)/handles.biphasic;
358         handles.duration_balance_total = handles.biphasic.*(handles.duration_1 +
        handles.duration_2);
359         handles.stimulation_fulltime = handles.duration_1+handles.
        duration_balance_total+handles.delay+handles.decay;
360         handles.fullstimulation_duration = handles.stimulation_fulltime;
361         handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
362         handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
363             (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
        time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
364             amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
365             zeros(1,(length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
366             -((handles.stimulation_fulltime)*handles.time_resolution))];
367         %
368         handles.current_data_I = handles.bi_level;
369         handles.stimulation_data_I = handles.bi_level;
370 
371         % Define which edit fields should work and define its initial values
372         set(handles.amplitude1_edit, 'enable', 'on','String', handles.amplitude_1
        )
373         set(handles.duration1_edit, 'enable', 'on','String', handles.duration_1)
374         set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
375         set(handles.delay_edit, 'enable', 'on','String', handles.delay)
376         set(handles.amplitude2_edit, 'enable', 'on','String', handles.amplitude_2
        )
377         set(handles.duration2_edit, 'enable', 'on','String', handles.duration_2 )
378         set(handles.decay_edit, 'enable', 'off')
379         set(handles.resolution_edit, 'enable', 'off')
380         set(handles.repetition_edit, 'enable', 'on','String', handles.repetition)
381         set(handles.Q_exp_edit, 'enable', 'off')
382         set(handles.Q2_exp_edit, 'enable', 'off')
383         set(handles.A_exp_edit, 'enable', 'off')
384         set(handles.A2_exp_edit, 'enable', 'off')
385         set(handles.B_exp_edit, 'enable', 'off')
386         set(handles.B2_exp_edit, 'enable', 'off')
387         set(handles.C_exp_edit, 'enable', 'off')
388         set(handles.C2_exp_edit, 'enable', 'off')
389         set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
```

```
390         set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
391
392     case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
393         handles.amplitude_1 = 1;
394         handles.duration_1 = 1;
395         handles.amplitude_2 = 0.5;
396         handles.duration_2 = 1;
397         handles.biphasic = 10;
398         handles.delay = 1;
399         handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
400         handles.duration_balance_2 = 0;
401         handles.decay = 1;
402         handles.repetition = 0;
403         handles.q = 1;
404         handles.q2 = 2.1972e+03;
405         handles.rep_amp_ratio = 1;
406
407         % Building the Quasi-Trapezoidal Wave
408         handles.decay_steps = handles.decay/handles.resolution;
409         handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
410         i=1;
411         total_decay_amplitude = 0;
412         for i=1:handles.decay_steps
413             handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.amplitude_1
    *handles.q*...
414                 exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s)).*...
415                 ones(1,handles.resolution*handles.time_resolution));
416             total_decay_amplitude = total_decay_amplitude + (handles.amplitude_1*
    handles.q*...
417                 exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
418         end
419         amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
420         handles.duration_balance_total = handles.biphasic.*(handles.duration_1 +
    handles.decay);
421         handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
422         handles.fullstimulation_duration = handles.stimulation_fulltime;
423         handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
424         handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.delay*
    handles.time_resolution))...
425             amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
426             zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
```

```
427             -((handles.stimulation_fulltime)*handles.time_resolution))]);
428         %
429         handles.current_data_I = handles.quasi_trap;
430         handles.stimulation_data_I = handles.quasi_trap;
431
432         % Define which edit fields should work and define its initial values
433         set(handles.amplitude1_edit, 'enable', 'on','String', handles.amplitude_1
    )
434         set(handles.duration1_edit, 'enable', 'on','String', handles.duration_1)
435         set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
436         set(handles.delay_edit, 'enable', 'on','String', handles.delay)
437         set(handles.amplitude2_edit, 'enable', 'off')
438         set(handles.duration2_edit, 'enable', 'off')
439         set(handles.decay_edit, 'enable', 'on','String', handles.decay)
440         set(handles.resolution_edit, 'enable', 'on','String', handles.resolution)
441         set(handles.repetition_edit, 'enable', 'on','String', handles.repetition)
442         set(handles.Q_exp_edit, 'enable', 'on','String', handles.q)
443         set(handles.Q2_exp_edit, 'enable', 'on', 'String', handles.q2)
444         set(handles.A_exp_edit, 'enable', 'off')
445         set(handles.A2_exp_edit, 'enable', 'off')
446         set(handles.B_exp_edit, 'enable', 'off')
447         set(handles.B2_exp_edit, 'enable', 'off')
448         set(handles.C_exp_edit, 'enable', 'off')
449         set(handles.C2_exp_edit, 'enable', 'off')
450         set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
451         set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
452
453     case 'Tri-Exponential' % User selects tri-exponential
454         handles.amplitude_1 = 1;
455         handles.duration_1 = 1;
456         handles.amplitude_2 = 0.5;
457         handles.duration_2 = 1;
458         handles.biphasic = 10;
459         handles.delay = 1;
460         handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
461         handles.duration_balance_2 = 0;
462         handles.decay = 1;
463         handles.repetition = 0;
464         handles.resolution = 0.25;
465         handles.a = 0.2;
466         handles.a2 = 1.197e+03;
467         handles.b = 0.3;
468         handles.b2 = 2.197e+03;
469         handles.c = 0.5;
470         handles.c2 = 3.197e+03;
471         handles.repetition = 0;
```

```
472         handles.rep_amp_ratio = 1;
473
474         % Building the Tri-Exponential Wave
475         handles.increase_steps = handles.duration_1/handles.resolution;
476         handles.decay_steps = handles.decay/handles.resolution;
477         handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
478         j_i = 1;
479         j_d = 1;
480         total_increase_amplitude = 0;
481         total_decay_amplitude = 0;
482         for j_i=1:handles.increase_steps
483             handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.amplitude_1)
    /((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.resolution*
    handles.time_resolution)...
484                 +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
485                 ones(1,handles.resolution*handles.time_resolution));
486             total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
487                 -((handles.amplitude_1*handles.t_i)/(handles.duration_1+handles.
    t_i)-handles.t_i))/handles.time_resolution);
488         end
489         for j_d=1:handles.decay_steps
490             handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((handles.
    t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
491                 exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))+(
    handles.b*...
492                 exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))+(
    handles.c*...
493                 exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s))))
    .*...
494                 ones(1,handles.resolution*handles.time_resolution));
495             total_decay_amplitude = total_decay_amplitude + (handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
496                 exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))+(
    handles.b*...
497                 exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))+(
    handles.c*...
498                 exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s))));
499         end
500         amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
501         handles.duration_balance_total = handles.biphasic.*(handles.duration_1 +
    handles.decay);
502         handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
503         handles.fullstimulation_duration = handles.stimulation_fulltime;
```

```
504         handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
505         handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
506             amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
507             zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
508             -((handles.stimulation_fulltime)*handles.time_resolution))]);
509         %
510         handles.current_data_I = handles.tri_exp;
511         handles.stimulation_data_I = handles.tri_exp;
512
513         % Define which edit fields should work and define its initial values
514         set(handles.amplitude1_edit, 'enable', 'on','String', handles.amplitude_1
    )
515         set(handles.duration1_edit, 'enable', 'on','String', handles.duration_1)
516         set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
517         set(handles.delay_edit, 'enable', 'on','String', handles.delay)
518         set(handles.amplitude2_edit, 'enable', 'off')
519         set(handles.duration2_edit, 'enable', 'off')
520         set(handles.decay_edit, 'enable', 'on','String', handles.decay)
521         set(handles.resolution_edit, 'enable', 'on','String', handles.resolution)
522         set(handles.repetition_edit, 'enable', 'on','String', handles.repetition)
523         set(handles.Q_exp_edit, 'enable', 'off')
524         set(handles.Q2_exp_edit, 'enable', 'off')
525         set(handles.A_exp_edit, 'enable', 'on','String', handles.a)
526         set(handles.A2_exp_edit, 'enable', 'on', 'String', handles.a2)
527         set(handles.B_exp_edit, 'enable', 'on','String', handles.b)
528         set(handles.B2_exp_edit, 'enable', 'on', 'String', handles.b2)
529         set(handles.C_exp_edit, 'enable', 'on','String', handles.c)
530         set(handles.C2_exp_edit, 'enable', 'on', 'String', handles.c2)
531         set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
532         set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
533
534     case 'Bursting' % User selects bursting strategies
535         handles.amplitude_1 = 1;
536         handles.duration_1 = 1;
537         handles.amplitude_2 = 0.5;
538         handles.duration_2 = 1;
539         handles.biphasic = 10;
540         handles.delay = 1;
541         handles.duration_balance_2 = 0;
542         handles.decay = 0;
543         handles.duration_balance_1 = handles.biphasic.*(handles.duration_1/2);
544         handles.resolution = 0.25;
```

```
545         handles.repetition = 0;
546         handles.rep_amp_ratio = 1;
547         handles.stimulation_fulltime = handles.duration_1+handles.duration_2+
        handles.duration_balance_1+handles.duration_balance_2+handles.delay+handles.
        decay;
548         handles.fullstimulation_duration = handles.stimulation_fulltime;
549         handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
550
551         % Building the Bursting Wave
552         handles.burst_steps = handles.duration_1/handles.resolution;
553         handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
554         k=1;
555         for k=1:handles.burst_steps
556             handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1
        .*...
557                 ones(1,(handles.resolution/2)*handles.time_resolution) ...
558                 zeros(1,(handles.resolution/2)*handles.time_resolution)]);
559         end
560         handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*
        handles.time_resolution))...
561             ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones(1,
        handles.duration_balance_1*handles.time_resolution)...
562             zeros(1,(length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
563             -((handles.stimulation_fulltime)*handles.time_resolution))]);
564         %
565         handles.current_data_I = handles.bursting;
566         handles.stimulation_data_I = handles.bursting;
567
568         % Define which edit fields should work and define its initial values
569         set(handles.amplitude1_edit, 'enable', 'on','String', handles.amplitude_1
        )
570         set(handles.duration1_edit, 'enable', 'on','String', handles.duration_1)
571         set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
572         set(handles.delay_edit, 'enable', 'on','String', handles.delay)
573         set(handles.amplitude2_edit, 'enable', 'off')
574         set(handles.duration2_edit, 'enable', 'off')
575         set(handles.decay_edit, 'enable', 'off')
576         set(handles.resolution_edit, 'enable', 'on','String', handles.resolution)
577         set(handles.repetition_edit, 'enable', 'on','String', handles.repetition)
578         set(handles.Q_exp_edit, 'enable', 'off')
579         set(handles.Q2_exp_edit, 'enable', 'off')
580         set(handles.A_exp_edit, 'enable', 'off')
581         set(handles.A2_exp_edit, 'enable', 'off')
582         set(handles.B_exp_edit, 'enable', 'off')
583         set(handles.B2_exp_edit, 'enable', 'off')
584         set(handles.C_exp_edit, 'enable', 'off')
```

```
585         set(handles.C2_exp_edit, 'enable', 'off')
586         set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
587         set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
588
589
590     case 'Load Waveform' % User selects to load an arbitrary waveform
591         final_answer = questdlg('Loaded Waveform can only have a maximum of 930
    data points.          Timings and Discharge Phase are of user responsability.
    Amplitude is a multiplication value in this case',...
592             'Load Waveform...','OK','Cancel','Cancel');
593         switch final_answer
594             case 'OK'
595                 handles.amplitude_1 = 1;
596                 handles.repetition = 0;
597                 handles.rep_amp_ratio = 1;
598
599                 % Define which edit fields should work and define its initial
    values
600                 set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
601                 set(handles.duration1_edit, 'enable', 'off')
602                 set(handles.amplitude2_edit, 'enable', 'off')
603                 set(handles.duration1_edit, 'enable', 'off')
604                 set(handles.duration2_edit, 'enable', 'off')
605                 set(handles.decay_edit, 'enable', 'off')
606                 set(handles.delay_edit, 'enable', 'off')
607                 set(handles.biphasic_edit, 'enable', 'off');
608                 set(handles.resolution_edit, 'enable', 'off')
609                 set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
610                 set(handles.Q_exp_edit, 'enable', 'off')
611                 set(handles.Q2_exp_edit, 'enable', 'off')
612                 set(handles.A_exp_edit, 'enable', 'off')
613                 set(handles.A2_exp_edit, 'enable', 'off')
614                 set(handles.B_exp_edit, 'enable', 'off')
615                 set(handles.B2_exp_edit, 'enable', 'off')
616                 set(handles.C_exp_edit, 'enable', 'off')
617                 set(handles.C2_exp_edit, 'enable', 'off')
618                 set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
619
620                 % Allows the use to choose the file to load
621                 try
622                     [FileName,PathName] = uigetfile('*.txt','Select the TXT file'
    );
623                     filename = [PathName FileName];
```

```
624                     delimiterIn = ' ';
625                     handles.current_data_I = importdata(filename,delimiterIn);
626                 catch err
627                     % Case the user doesnt want to load waveform afterall
628                     errordlg('Something went wrong',...
629                         'Invalid Input','modal')
630                     uicontrol(hObject)
631                     return
632                 end
633 
634                 % Checks is data has the limit size
635                 if length(handles.current_data_I) > 930
636                     errordlg('Too many data points!','Invalid TXT File','modal')
637                     uicontrol(hObject)
638                     return
639                 else
640                     handles.stimulation_data_I = handles.current_data_I;
641                     set(handles.stim_duration_edit, 'enable', 'on','String',
    (0:(1/handles.time_resolution):(lenght(handles.current_data_I)/handles.
    time_resolution)))
642                 end
643             case ' Cancel'
644         end
645 
646 end
647 
648 guidata(hObject, handles);
649 
650 
651 % --- Executes during object creation, after setting all properties.
652 function plot_popup_CreateFcn(hObject, eventdata, handles)
653 % hObject    handle to plot_popup (see GCBO)
654 % eventdata  reserved - to be defined in a future version of MATLAB
655 % handles    empty - handles not created until after all CreateFcns called
656 
657 % Hint: popupmenu controls usually have a white background on Windows.
658 %       See ISPC and COMPUTER.
659 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
660     set(hObject,'BackgroundColor','white');
661 end
662 
663 
664 
665 function amplitude1_edit_Callback(hObject, eventdata, handles)
666 % hObject    handle to amplitude1_edit (see GCBO)
667 % eventdata  reserved - to be defined in a future version of MATLAB
668 % handles    structure with handles and user data (see GUIDATA)
```

```
669
670 % Hints: get(hObject,'String') returns contents of amplitude1_edit as text
671 %        str2double(get(hObject,'String')) returns contents of amplitude1_edit as
        a double
672
673 % Data input by user
674 amplitude_1_input = str2double(get(hObject,'string'));
675
676 if isfield(handles, 'plot_popup_value' )
677     % handles.plot_popup_value exists - you may access it
678 else
679     % handles.plot_popup_value does not exist
680     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
681     uicontrol(hObject)
682     return
683 end
684
685 % If there is no amplitude input, if it is non-numeric, or if the input is 0,
        error
686 if isnan(amplitude_1_input) || amplitude_1_input == 0
687     errordlg('You must enter a numeric value bigger than 0 (zero)','Invalid Input
        ','modal')
688     uicontrol(hObject)
689     return
690 % If there is a maximum amplitude calculated, it confirms if its below it
        otherwise error
691 elseif handles.amplitude_max ~=0 && amplitude_1_input > handles.amplitude_max
692     errordlg('That value breaches the Safe Limit Range','Invalid Input','modal')
693     uicontrol(hObject)
694     return
695 % The Quad-DAC maximum Output is 5V
696 elseif amplitude_1_input > 5
697     errordlg('DACs higher output is 5V','Invalid Input','modal')
698     uicontrol(hObject)
699     return
700 % If the value doesnt have any 'problem', builds the Waveforms with the
701 % inputed value
702 else
703     handles.amplitude_1 = amplitude_1_input;
704     value = handles.plot_popup_value;
705     string = handles.plot_popup_string;
706     switch string{value}
707         case 'Bi-level Lilly' % Case user has selected bilevel Lilly
708             amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
        amplitude_2*handles.duration_2)/handles.biphasic;
709             handles.duration_balance_total = handles.biphasic.*(handles.
        duration_1 + handles.duration_2);
```

```
710             handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
711             handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
712             handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
713                 (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
    time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
714                 amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
715                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
716                 -((handles.stimulation_fulltime)*handles.time_resolution)))];
717             handles.current_data_I = handles.bi_level;
718             handles.stimulation_data_I = handles.bi_level;
719         case 'Quasi-Trapezoidal' % Case user has selected quasi-trapezoidal
720             handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
721             i=1;
722             total_decay_amplitude = 0;
723             for i=1:handles.decay_steps
724                 handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
    amplitude_1*handles.q*...
725                     exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
    .*...
726                     ones(1,handles.resolution*handles.time_resolution));
727                 total_decay_amplitude = total_decay_amplitude + (handles.
    amplitude_1*handles.q*...
728                     exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
729             end
730             amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
731             handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
732             handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
733             handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
734             handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
735                 amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
736                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
737                 -((handles.stimulation_fulltime)*handles.time_resolution)))]);
738             handles.current_data_I = handles.quasi_trap;
739             handles.stimulation_data_I = handles.quasi_trap;
740         case 'Tri-Exponential' % Case user has selected tri-exponential
```

```
741             handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
742             j_i = 1;
743             j_d = 1;
744             total_increase_amplitude = 0;
745             total_decay_amplitude = 0;
746             for j_i=1:handles.increase_steps
747                 handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
    resolution*handles.time_resolution)...
748                     +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
749                     ones(1,handles.resolution*handles.time_resolution));
750                 total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
751                     -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
752             end
753             for j_d=1:handles.decay_steps
754                 handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
755                     exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
756                     exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
757                     exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
758                     ones(1,handles.resolution*handles.time_resolution));
759                 total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
760                     exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
761                     exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
762                     exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
763             end
764             amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
765             handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
766             handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
767             handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
768             handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
```

```
769                 amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
770                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
771                 -((handles.stimulation_fulltime)*handles.time_resolution)))]);
772             handles.current_data_I = handles.tri_exp;
773             handles.stimulation_data_I = handles.tri_exp;
774         case 'Bursting' % Case user has selected bursting strategies
775             handles.duration_balance_1 = handles.biphasic.*(handles.duration_1/2)
    ;
776             handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
777             k=1;
778             for k=1:handles.burst_steps
779                 handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1
    .*...
780                     ones(1,(handles.resolution/2)*handles.time_resolution) ...
781                     zeros(1,(handles.resolution/2)*handles.time_resolution)]);
782             end
783             handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*
    handles.time_resolution))...
784                 ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
785                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
786                 -((handles.stimulation_fulltime)*handles.time_resolution)))]);
787             handles.current_data_I = handles.bursting;
788             handles.stimulation_data_I = handles.bursting;
789         case 'Rectangular' % Case user has selected pulse
790             handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
791                 zeros(1,(handles.delay*handles.time_resolution))...
792                 ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
793                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
794                 -((handles.stimulation_fulltime)*handles.time_resolution)))];
795             handles.current_data_I = handles.pulse;
796             handles.stimulation_data_I = handles.pulse;
797         case 'Load Waveform' % Case user has loaded its own waveform
798             handles.current_data_I = handles.amplitude_1.* handles.current_data_I
    ;
799             handles.stimulation_data_I = handles.current_data_I;
800     end
801
802     % If there is a repetition defined, it rebuilds the waveform with the
803     % repetitions
804     handles.reps_max = 0;
805     if handles.repetition ~= 0
```

```
806         for n_reps = 1:handles.n_repetitions
807             if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
        amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
         handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
        reps_max == 0)...
808                     || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
         <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
        DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
        +1)) ) && handles.reps_max == 0)
809                 handles.reps_max = n_reps;
810             end
811             if handles.reps_max ~=0
812                 handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
        handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
        .^(handles.reps_max)).*handles.stimulation_data_I);
813             else
814                 handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
        handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
        .^(n_reps)).*handles.stimulation_data_I);
815             end
816         end
817     end
818
819     % If time stimulation created is bigger than the chosen value of stimulation,
         it
820     % replaces it on the GUI
821     if handles.fullstimulation_duration < handles.stimulation_fulltime
822         set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
823     end
824
825 end
826
827 guidata(hObject, handles);
828
829
830 % --- Executes during object creation, after setting all properties.
831 function amplitude1_edit_CreateFcn(hObject, eventdata, handles)
832 % hObject    handle to amplitude1_edit (see GCBO)
833 % eventdata  reserved - to be defined in a future version of MATLAB
834 % handles    empty - handles not created until after all CreateFcns called
835
836 % Hint: edit controls usually have a white background on Windows.
837 %       See ISPC and COMPUTER.
838 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
839     set(hObject,'BackgroundColor','white');
840 end
```

```
841
842
843
844 function duration1_edit_Callback(hObject, eventdata, handles)
845 % hObject    handle to duration1_edit (see GCBO)
846 % eventdata  reserved - to be defined in a future version of MATLAB
847 % handles    structure with handles and user data (see GUIDATA)
848
849 % Hints: get(hObject,'String') returns contents of duration1_edit as text
850 %        str2double(get(hObject,'String')) returns contents of duration1_edit as
        a double
851
852 % Data input by user
853 duration_1_input = str2double(get(hObject,'string'));
854
855 if isfield(handles, 'plot_popup_value' )
856     % handles.plot_popup_value exists - you may access it
857 else
858     % handles.plot_popup_value does not exist
859     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
860     uicontrol(hObject)
861     return
862 end
863
864 % If there is no duration value, if it is non-numeric, or if it is smaller than
        0.1ms, error
865 if isnan(duration_1_input) || duration_1_input < 0.1
866     errordlg('You must enter a numeric value bigger than 0.1','Invalid Input','
        modal')
867     uicontrol(hObject)
868     return
869 % If the input value leads to a non-integer value when multiplied by the
870 % time resolution  it throws error because it would lead to a array error
871 elseif ((duration_1_input*handles.time_resolution) - floor(duration_1_input*
        handles.time_resolution)) ~= 0
872     errordlg('Invalid input','Invalid Input','modal')
873     uicontrol(hObject)
874     return
875 % If the length of the one pulse stimulation is bigger than the MCU maximum,
876 % do to its memory limitations, error
877 elseif length((0:(1/handles.time_resolution):(duration_1_input+handles.duration_2
        +(handles.biphasic*duration_1_input)+(handles.biphasic*handles.decay)+(
        handles.biphasic*handles.duration_2)+handles.delay+handles.decay))) > 930
878     errordlg('The full duration of the pulse has to be smaller','Invalid Input','
        modal')
879     uicontrol(hObject)
880     return
881 % Else, it starts building the waveform
```

```
882 else
883     handles.duration_1 = duration_1_input;
884     handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
885     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
        duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
886     handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
887     handles.burst_steps = handles.duration_1/handles.resolution;
888     handles.increase_steps = handles.duration_1/handles.resolution;
889     value = handles.plot_popup_value;
890     string = handles.plot_popup_string;
891     switch string{value}
892         case 'Bi-level Lilly' % Case user has selected bilevel Lilly
893             amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
        amplitude_2*handles.duration_2)/handles.biphasic;
894             handles.duration_balance_total = handles.biphasic.*(handles.
        duration_1 + handles.duration_2);
895             handles.stimulation_fulltime = handles.duration_1+handles.
        duration_balance_total+handles.delay+handles.decay;
896             handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
897             handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
898                 (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
        time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
899                 amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
900                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
901                 -((handles.stimulation_fulltime)*handles.time_resolution)))];
902             handles.current_data_I = handles.bi_level;
903             handles.stimulation_data_I = handles.bi_level;
904         case 'Quasi-Trapezoidal' % Case user has selected quasi-trapezoidal
905             handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
906             i=1;
907             total_decay_amplitude = 0;
908             for i=1:handles.decay_steps
909                 handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
        amplitude_1*handles.q*...
910                     exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
        .*...
911                     ones(1,handles.resolution*handles.time_resolution));
912                 total_decay_amplitude = total_decay_amplitude + (handles.
        amplitude_1*handles.q*...
913                     exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
914             end
```

```
915             amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
916             handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
917             handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
918             handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
919             handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
920                 amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
921                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
922                 -((handles.stimulation_fulltime)*handles.time_resolution)))]);
923             handles.current_data_I = handles.quasi_trap;
924             handles.stimulation_data_I = handles.quasi_trap;
925         case 'Tri-Exponential' % Case user has selected tri-exponential
926             handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
927             j_i = 1;
928             j_d = 1;
929             total_increase_amplitude = 0;
930             total_decay_amplitude = 0;
931             for j_i=1:handles.increase_steps
932                 handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
    resolution*handles.time_resolution)...
933                     +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
934                     ones(1,handles.resolution*handles.time_resolution));
935                 total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
936                     -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
937             end
938             for j_d=1:handles.decay_steps
939                 handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
940                     exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
941                     exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
942                     exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
943                     ones(1,handles.resolution*handles.time_resolution));
944                 total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
```

```
945                     exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
946                     exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
947                     exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
948             end
949             amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
950             handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
951             handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
952             handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
953             handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
954                 amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
955                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
956                 -((handles.stimulation_fulltime)*handles.time_resolution)))]);
957             handles.current_data_I = handles.tri_exp;
958             handles.stimulation_data_I = handles.tri_exp;
959         case 'Bursting' % Case user has selected bursting strategies
960             handles.duration_balance_1 = handles.biphasic.*(handles.duration_1/2)
    ;
961             handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
962             k=1;
963             for k=1:handles.burst_steps
964                 handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1
    .*...
965                     ones(1,(handles.resolution/2)*handles.time_resolution) ...
966                     zeros(1,(handles.resolution/2)*handles.time_resolution)]);
967             end
968             handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*
    handles.time_resolution))...
969                 ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
970                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
971                 -((handles.stimulation_fulltime)*handles.time_resolution)))]);
972             handles.current_data_I = handles.bursting;
973             handles.stimulation_data_I = handles.bursting;
974         case 'Rectangular' % Case user has selected pulse
975             handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
976                 zeros(1,(handles.delay*handles.time_resolution))...
```

```
977                 ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
978                 zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
979                 -((handles.stimulation_fulltime)*handles.time_resolution)))];
980             handles.current_data_I = handles.pulse;
981             handles.stimulation_data_I = handles.pulse;
982     end
983
984     % If there is a repetition defined, it rebuilds the waveform with the
985     % repetitions
986     handles.reps_max = 0;
987     if handles.repetition ~= 0
988         for n_reps = 1:handles.n_repetitions
989             if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
990                     || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
991                 handles.reps_max = n_reps;
992             end
993             if handles.reps_max ~=0
994                 handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
995             else
996                 handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
997             end
998         end
999     end
1000
1001    % If time stimulation created is bigger than the chosen value of stimulation,
     it
1002    % replaces it on the GUI
1003    if handles.fullstimulation_duration < handles.stimulation_fulltime
1004        set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
1005    end
1006
1007 end
1008
1009
1010 guidata(hObject, handles);
```

```
1011
1012
1013 % --- Executes during object creation, after setting all properties.
1014 function duration1_edit_CreateFcn(hObject, eventdata, handles)
1015 % hObject    handle to duration1_edit (see GCBO)
1016 % eventdata  reserved - to be defined in a future version of MATLAB
1017 % handles    empty - handles not created until after all CreateFcns called
1018
1019 % Hint: edit controls usually have a white background on Windows.
1020 %       See ISPC and COMPUTER.
1021 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
         defaultUicontrolBackgroundColor'))
1022     set(hObject,'BackgroundColor','white');
1023 end
1024
1025
1026
1027 function delay_edit_Callback(hObject, eventdata, handles)
1028 % hObject    handle to delay_edit (see GCBO)
1029 % eventdata  reserved - to be defined in a future version of MATLAB
1030 % handles    structure with handles and user data (see GUIDATA)
1031
1032 % Hints: get(hObject,'String') returns contents of delay_edit as text
1033 %        str2double(get(hObject,'String')) returns contents of delay_edit as a
         double
1034
1035 % Data input by user
1036 delay_input = str2double(get(hObject,'string'));
1037
1038 if isfield(handles, 'plot_popup_value' )
1039     % handles.firing_popup_value exists - you may access it
1040 else
1041     % handles.firing_popup_value does not exist
1042     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
1043     uicontrol(hObject)
1044     return
1045 end
1046
1047 % If there is no delay value or has a non-number value, error
1048 if isnan(delay_input)
1049     errordlg('You must enter a numeric value','Invalid Input','modal')
1050     uicontrol(hObject)
1051     return
1052 % If the input value leads to a non-integer value when multiplied by the
1053 % time resolution  it throws error because it would lead to a array error
1054 elseif ((delay_input*handles.time_resolution) - floor(delay_input*handles.
         time_resolution)) ~= 0
1055     errordlg('Invalid input','Invalid Input','modal')
```

```
1056        uicontrol(hObject)
1057        return
1058    % If the length of the one pulse stimulation is bigger than the MCU maximum,
1059    % do to its memory limitations, error
1060    elseif length((0:(1/handles.time_resolution):(handles.duration_1+handles.
            duration_2+(handles.biphasic*(handles.duration_1 + handles.duration_2+handles
            .delay))+delay_input+handles.decay))) > 930
1061        errordlg('The full duration of the pulse has to be smaller','Invalid Input','
            modal')
1062        uicontrol(hObject)
1063        return
1064    % Else, waveform is build
1065    else
1066        handles.delay = delay_input;
1067        handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
            duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
1068        handles.timescale = (0:(1/handles.time_resolution):(handles.
            stimulation_fulltime+handles.t_f));
1069        handles.duration_balance_1 = handles.biphasic*handles.duration_1;
1070        value = handles.plot_popup_value;
1071        string = handles.plot_popup_string;
1072        switch string{value}
1073            case 'Bi-level Lilly' % User selects bilevel Lilly
1074                amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
            amplitude_2*handles.duration_2)/handles.biphasic;
1075                handles.duration_balance_total = handles.biphasic.*(handles.
            duration_1 + handles.duration_2);
1076                handles.stimulation_fulltime = handles.duration_1+handles.
            duration_balance_total+handles.delay+handles.decay;
1077                handles.timescale = (0:(1/handles.time_resolution):(handles.
            stimulation_fulltime+handles.t_f));
1078                handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
            handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
1079                    (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
            time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
1080                    amplitude_CB.*ones(1,handles.duration_balance_total*handles.
            time_resolution)...
1081                    zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
            time_resolution)...
1082                    -((handles.stimulation_fulltime)*handles.time_resolution)))];
1083                handles.current_data_I = handles.bi_level;
1084                handles.stimulation_data_I = handles.bi_level;
1085            case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
1086                handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
            handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
1087                i=1;
1088                total_decay_amplitude = 0;
1089                for i=1:handles.decay_steps
```

```
1090                handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
    amplitude_1*handles.q*...
1091                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
    .*...
1092                    ones(1,handles.resolution*handles.time_resolution));
1093                total_decay_amplitude = total_decay_amplitude + (handles.
    amplitude_1*handles.q*...
1094                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
1095            end
1096            amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
1097            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
1098            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
1099            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
1100            handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
1101                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
1102                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1103                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1104            handles.current_data_I = handles.quasi_trap;
1105            handles.stimulation_data_I = handles.quasi_trap;
1106        case 'Tri-Exponential' % User selects tri-exponential
1107            handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
1108            j_i = 1;
1109            j_d = 1;
1110            total_increase_amplitude = 0;
1111            total_decay_amplitude = 0;
1112            for j_i=1:handles.increase_steps
1113                handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
    resolution*handles.time_resolution)...
1114                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
1115                    ones(1,handles.resolution*handles.time_resolution));
1116                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
1117                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
1118            end
1119            for j_d=1:handles.decay_steps
1120                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
```

```
1121                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
        +(handles.b*...
1122                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
        +(handles.c*...
1123                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
        ).*...
1124                    ones(1,handles.resolution*handles.time_resolution));
1125                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
        ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
1126                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
        +(handles.b*...
1127                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
        +(handles.c*...
1128                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
        );
1129            end
1130            amplitude_CB = (total_increase_amplitude*handles.resolution +
        total_decay_amplitude*handles.resolution)/handles.biphasic;
1131            handles.duration_balance_total = handles.biphasic.*(handles.
        duration_1 + handles.decay);
1132            handles.stimulation_fulltime = handles.duration_1+handles.
        duration_balance_total+handles.delay+handles.decay;
1133            handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
1134            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
        handles.time_resolution))...
1135                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
1136                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
1137                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1138            handles.current_data_I = handles.tri_exp;
1139            handles.stimulation_data_I = handles.tri_exp;
1140        case 'Bursting' % User selects bursting strategies
1141            handles.duration_balance_1 = handles.biphasic.*(handles.duration_1/2)
        ;
1142            handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
1143            k=1;
1144            for k=1:handles.burst_steps
1145                handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1
        .*...
1146                    ones(1,(handles.resolution/2)*handles.time_resolution) ...
1147                    zeros(1,(handles.resolution/2)*handles.time_resolution)]);
1148            end
1149            handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*
        handles.time_resolution))...
1150                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
        (1,handles.duration_balance_1*handles.time_resolution)...
```

```
1151                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1152                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1153            handles.current_data_I = handles.bursting;
1154            handles.stimulation_data_I = handles.bursting;
1155        case 'Rectangular' % User selects squar
1156            handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
1157                zeros(1,(handles.delay*handles.time_resolution))...
1158                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
1159                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1160                -((handles.stimulation_fulltime)*handles.time_resolution)))];
1161            handles.current_data_I = handles.pulse;
1162            handles.stimulation_data_I = handles.pulse;
1163    end
1164
1165    % If there is a repetition defined, it rebuilds the waveform with the
1166    % repetitions
1167    handles.reps_max = 0;
1168    if handles.repetition ~= 0
1169        for n_reps = 1:handles.n_repetitions
1170            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
1171                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
1172                handles.reps_max = n_reps;
1173            end
1174            if handles.reps_max ~=0
1175                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
1176            else
1177                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
1178            end
1179        end
1180    end
1181
1182    % If time stimulation created is bigger than the chosen value of stimulation,
     it
1183    % replaces it on the GUI
```

```
1184      if handles.fullstimulation_duration < handles.stimulation_fulltime
1185          set(handles.stim_duration_edit, 'enable', 'on','String', handles.
          stimulation_fulltime)
1186      end
1187
1188  end
1189
1190  guidata(hObject, handles);
1191
1192
1193  % --- Executes during object creation, after setting all properties.
1194  function delay_edit_CreateFcn(hObject, eventdata, handles)
1195  % hObject    handle to delay_edit (see GCBO)
1196  % eventdata  reserved - to be defined in a future version of MATLAB
1197  % handles    empty - handles not created until after all CreateFcns called
1198
1199  % Hint: edit controls usually have a white background on Windows.
1200  %       See ISPC and COMPUTER.
1201  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
          defaultUicontrolBackgroundColor'))
1202      set(hObject,'BackgroundColor','white');
1203  end
1204
1205
1206
1207  function biphasic_edit_Callback(hObject, eventdata, handles)
1208  % hObject    handle to biphasic_edit (see GCBO)
1209  % eventdata  reserved - to be defined in a future version of MATLAB
1210  % handles    structure with handles and user data (see GUIDATA)
1211
1212  % Hints: get(hObject,'String') returns contents of biphasic_edit as text
1213  %        str2double(get(hObject,'String')) returns contents of biphasic_edit as a
           double
1214
1215  % Data input by user
1216  biphasic_input = str2double(get(hObject,'string'));
1217
1218  if isfield(handles, 'plot_popup_value' )
1219      % handles.firing_popup_value exists - you may access it
1220  else
1221      % handles.firing_popup_value does not exist
1222      errordlg('You must choose a Waverform Type!','Invalid Input','modal')
1223      uicontrol(hObject)
1224      return
1225  end
1226
1227  % If there is no biphasic value or its non-numeric
1228  if isnan(biphasic_input) %|| biphasic_input == 0
```

```
1229        errordlg('You must enter a numeric value','Invalid Input','modal')
1230        uicontrol(hObject)
1231        return
1232    elseif length((0:(1/handles.time_resolution):(handles.duration_1+handles.
        duration_2+(biphasic_input*handles.duration_1)+(biphasic_input*handles.
        duration_2)+(biphasic_input*handles.decay)+handles.delay+handles.decay))) >
        930
1233        errordlg('The full duration of the pulse has to be smaller','Invalid Input','
        modal')
1234        uicontrol(hObject)
1235        return
1236    else
1237        handles.biphasic = biphasic_input;
1238        handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
1239        handles.duration_balance_2 = handles.biphasic*handles.duration_2;
1240        handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
        duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
1241        handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
1242        value = handles.plot_popup_value;
1243        string = handles.plot_popup_string;
1244        switch string{value}
1245            case 'Bi-level Lilly' % User selects bilevel Lilly
1246                amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
        amplitude_2*handles.duration_2)/handles.biphasic;
1247                handles.duration_balance_total = handles.biphasic.*(handles.
        duration_1 + handles.duration_2);
1248                handles.stimulation_fulltime = handles.duration_1+handles.
        duration_balance_total+handles.delay+handles.decay;
1249                handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
1250                handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
1251                    (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
        time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
1252                    amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
1253                    zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
1254                    -((handles.stimulation_fulltime)*handles.time_resolution)))];
1255                handles.current_data_I = handles.bi_level;
1256                handles.stimulation_data_I = handles.bi_level;
1257            case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
1258                handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
1259                i=1;
1260                total_decay_amplitude = 0;
1261                for i=1:handles.decay_steps
```

```
1262                handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
    amplitude_1*handles.q*...
1263                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
    .*...
1264                    ones(1,handles.resolution*handles.time_resolution));
1265                total_decay_amplitude = total_decay_amplitude + (handles.
    amplitude_1*handles.q*...
1266                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
1267            end
1268            amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
1269            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
1270            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
1271            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
1272            handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
1273                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
1274                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1275                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1276            handles.current_data_I = handles.quasi_trap;
1277            handles.stimulation_data_I = handles.quasi_trap;
1278        case 'Tri-Exponential' % User selects tri-exponential
1279            handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
1280            j_i = 1;
1281            j_d = 1;
1282            total_increase_amplitude = 0;
1283            total_decay_amplitude = 0;
1284            for j_i=1:handles.increase_steps
1285                handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
    resolution*handles.time_resolution)...
1286                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
1287                    ones(1,handles.resolution*handles.time_resolution));
1288                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
1289                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
1290            end
1291            for j_d=1:handles.decay_steps
1292                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
```

```
1293                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
1294                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
1295                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
1296                    ones(1,handles.resolution*handles.time_resolution));
1297                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
1298                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
1299                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
1300                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
1301            end
1302            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
1303            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
1304            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
1305            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
1306            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
1307                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
1308                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1309                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1310            handles.current_data_I = handles.tri_exp;
1311            handles.stimulation_data_I = handles.tri_exp;
1312        case 'Bursting' % User selects bursting strategies
1313            handles.duration_balance_1 = handles.biphasic.*(handles.duration_1/2)
    ;
1314            handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
1315            k=1;
1316            for k=1:handles.burst_steps
1317                handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1
    .*...
1318                    ones(1,(handles.resolution/2)*handles.time_resolution) ...
1319                    zeros(1,(handles.resolution/2)*handles.time_resolution)]);
1320            end
1321            handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*
    handles.time_resolution))...
1322                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
```

```
1323                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1324                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1325            handles.current_data_I = handles.bursting;
1326            handles.stimulation_data_I = handles.bursting;
1327        case 'Rectangular' % User selects square
1328            handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
1329                zeros(1,(handles.delay*handles.time_resolution))...
1330                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
1331                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1332                -((handles.stimulation_fulltime)*handles.time_resolution)))];
1333            handles.current_data_I = handles.pulse;
1334            handles.stimulation_data_I = handles.pulse;
1335    end
1336
1337    % If there is a repetition defined, it rebuilds the waveform with the
1338    % repetitions
1339    handles.reps_max = 0;
1340    if handles.repetition ~= 0
1341        for n_reps = 1:handles.n_repetitions
1342            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
1343                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
1344                handles.reps_max = n_reps;
1345            end
1346            if handles.reps_max ~=0
1347                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
1348            else
1349                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
1350            end
1351        end
1352    end
1353
1354    % If time stimulation created is bigger than the chosen value of stimulation,
     it
1355    % replaces it on the GUI
```

```
1356      if handles.fullstimulation_duration < handles.stimulation_fulltime
1357          set(handles.stim_duration_edit, 'enable', 'on','String', handles.
          stimulation_fulltime)
1358      end
1359
1360  end
1361
1362  set(handles.stim_duration_edit, 'enable', 'on','String', handles.
          stimulation_fulltime)
1363
1364
1365  guidata(hObject, handles);
1366
1367
1368  % --- Executes during object creation, after setting all properties.
1369  function biphasic_edit_CreateFcn(hObject, eventdata, handles)
1370  % hObject    handle to biphasic_edit (see GCBO)
1371  % eventdata  reserved - to be defined in a future version of MATLAB
1372  % handles    empty - handles not created until after all CreateFcns called
1373
1374  % Hint: edit controls usually have a white background on Windows.
1375  %       See ISPC and COMPUTER.
1376  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
          defaultUicontrolBackgroundColor'))
1377      set(hObject,'BackgroundColor','white');
1378  end
1379
1380
1381
1382  function repetition_edit_Callback(hObject, eventdata, handles)
1383  % hObject    handle to repetition_edit (see GCBO)
1384  % eventdata  reserved - to be defined in a future version of MATLAB
1385  % handles    structure with handles and user data (see GUIDATA)
1386
1387  % Hints: get(hObject,'String') returns contents of repetition_edit as text
1388  %        str2double(get(hObject,'String')) returns contents of repetition_edit as
           a double
1389
1390  % Data input by user
1391  repetition_input = str2double(get(hObject,'string'));
1392
1393  if isfield(handles, 'plot_popup_value' )
1394      % handles.firing_popup_value exists - you may access it
1395  else
1396      % handles.firing_popup_value does not exist
1397      errordlg('You must choose a Waverform Type!','Invalid Input','modal')
1398      uicontrol(hObject)
1399      return
```

```
1400 end
1401
1402 % If there is no repetition value or if it is bigger than 40kHz
1403 if isnan(repetition_input) || repetition_input > 40
1404     errordlg('You must enter a numeric value smaller or equal to 40kHz','Invalid
         Input','modal')
1405     uicontrol(hObject)
1406     return
1407 % If repetition input is a decimal number not compatible with time
1408 % resolution
1409 elseif rem(handles.time_resolution,repetition_input) ~= 0 && rem(repetition_input
         ,handles.time_resolution) ~= 0
1410     errordlg('Invalid input','Invalid Input','modal')
1411     uicontrol(hObject)
1412     return
1413 % If input is 0, waveform is build with no repetition
1414 elseif repetition_input == 0
1415     handles.interstimulus_time = 0;
1416     handles.n_repetitions = 1;
1417     value = handles.plot_popup_value;
1418     string = handles.plot_popup_string;
1419     switch string{value}
1420         case 'Bi-level Lilly' % User selects bilevel Lilly
1421             handles.current_data_I = handles.bi_level;
1422             handles.stimulation_data_I = handles.bi_level;
1423         case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
1424             handles.current_data_I = handles.quasi_trap;
1425             handles.stimulation_data_I = handles.quasi_trap;
1426         case 'Tri-Exponential' % User selects tri-exponential
1427             handles.current_data_I = handles.tri_exp;
1428             handles.stimulation_data_I = handles.tri_exp;
1429         case 'Bursting' % User selects bursting strategies
1430             handles.current_data_I = handles.bursting;
1431             handles.stimulation_data_I = handles.bursting;
1432         case 'Rectangular' % User selects pulse
1433             handles.current_data_I = handles.pulse;
1434             handles.stimulation_data_I = handles.pulse;
1435         case 'Load Waveform'
1436             handles.current_data_I = handles.current_data_I;
1437             handles.stimulation_data_I = handles.current_data_I;
1438     end
1439     return
1440 % If interstimulus time is smaller than stimulation duration, no repetition
1441 % is made
1442 elseif ((1/(repetition_input)) > handles.fullstimulation_duration)
1443     warndlg('Repetition Cycle bigger than Stimulation Duration! No Repetition
         will be made.','Value Selection','modal');
1444     uicontrol(hObject)
```

```
1445      return
1446  % Else, build waveform with the calculated number of repetitions
1447  % The builted waveform is just for the plot, since the MATLAB sence only
1448  % one pulse and the information on no of repetions and interstimulus time
1449  else
1450      handles.repetition = repetition_input;
1451      value = handles.plot_popup_value;
1452      string = handles.plot_popup_string;
1453      n_reps = 1;
1454      handles.reps_max = 0;
1455      handles.interstimulus_time = 1/handles.repetition;
1456      handles.n_repetitions = (floor(handles.fullstimulation_duration/(handles.
         stimulation_fulltime+handles.interstimulus_time))-1);
1457      switch string{value}
1458          case 'Bi-level Lilly' % User selects bilevel Lilly
1459              handles.current_data_I = handles.bi_level;
1460              handles.stimulation_data_I = handles.bi_level;
1461              for n_reps = 1:handles.n_repetitions
1462                  if (handles.amplitude_max ~=0 && handles.reps_max == 0 && (
         handles.amplitude_max <= handles.amplitude_1*((handles.rep_amp_ratio).^(
         n_reps+1)) || handles.amplitude_max <= handles.amplitude_2*((handles.
         rep_amp_ratio).^(n_reps+1))))...
1463                          || (handles.amplitude_max ==0 && (handles.
         DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
         +1)) || handles.DAC_amplitude_limit <= handles.amplitude_2*((handles.
         rep_amp_ratio).^(n_reps+1)) ) && handles.reps_max == 0)
1464                      handles.reps_max = n_reps;
1465                  end
1466                  if handles.reps_max ~=0
1467                      handles.current_data_I = horzcat(handles.current_data_I,zeros
         (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
         rep_amp_ratio).^(handles.reps_max)).*handles.bi_level);
1468                  else
1469                      handles.current_data_I = horzcat(handles.current_data_I,zeros
         (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
         rep_amp_ratio).^(n_reps)).*handles.bi_level);
1470                  end
1471              end
1472          case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
1473              handles.current_data_I = handles.quasi_trap;
1474              handles.stimulation_data_I = handles.quasi_trap;
1475              for n_reps = 1:handles.n_repetitions
1476                  if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
         .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
         ...
1477                          || (handles.amplitude_max ==0 && handles.
         DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
         +1)) && handles.reps_max == 0)
```

```
1478                    handles.reps_max = n_reps;
1479                end
1480                if handles.reps_max ~=0
1481                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.quasi_trap);
1482                else
1483                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.quasi_trap);
1484                end
1485            end
1486        case 'Tri-Exponential' % User selects tri-exponential
1487            handles.current_data_I = handles.tri_exp;
1488            handles.stimulation_data_I = handles.tri_exp;
1489            for n_reps = 1:handles.n_repetitions
1490                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
1491                        || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
1492                    handles.reps_max = n_reps;
1493                end
1494                if handles.reps_max ~=0
1495                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.tri_exp);
1496                else
1497                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.tri_exp);
1498                end
1499            end
1500        case 'Bursting' % User selects bursting strategies
1501            handles.current_data_I = handles.bursting;
1502            handles.stimulation_data_I = handles.bursting;
1503            for n_reps = 1:handles.n_repetitions
1504                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
1505                        || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
1506                    handles.reps_max = n_reps;
1507                end
1508                if handles.reps_max ~=0
```

```
1509                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.bursting);
1510                else
1511                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.bursting);
1512                end
1513            end
1514        case 'Rectangular' % User selects pulse
1515            handles.current_data_I = handles.pulse;
1516            handles.stimulation_data_I = handles.pulse;
1517            for n_reps = 1:handles.n_repetitions
1518                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
1519                        || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
1520                    handles.reps_max = n_reps;
1521                end
1522                if handles.reps_max ~=0
1523                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.pulse);
1524                else
1525                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.pulse);
1526                end
1527            end
1528        case 'Load Waveform'
1529            handles.current_data_I = handles.current_data_I;
1530            handles.stimulation_data_I = handles.current_data_I;
1531            for n_reps = 1:handles.n_repetitions
1532                if (handles.amplitude_max ~=0 && (handles.amplitude_max <= max(
    handles.stimulation_data_I)*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    amplitude_max <= abs(min(handles.stimulation_data_I))*((handles.rep_amp_ratio
    ).^(n_reps+1))) && handles.reps_max == 0)...
1533                        || (handles.amplitude_max ==0 && (handles.
    DAC_amplitude_limit <= max(handles.stimulation_data_I)*((handles.
    rep_amp_ratio).^(n_reps+1)) || handles.DAC_amplitude_limit <= abs(min(handles
    .stimulation_data_I))*((handles.rep_amp_ratio).^(n_reps+1))) && handles.
    reps_max == 0)
1534                    handles.reps_max = n_reps;
1535                end
1536                if handles.reps_max ~=0
```

```
1537                        handles.current_data_I = horzcat(handles.current_data_I,zeros
        (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
        rep_amp_ratio).^(n_reps)).*handles.stimulation_data_I);
1538                    else
1539                        handles.current_data_I = horzcat(handles.current_data_I,zeros
        (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
        rep_amp_ratio).^(n_reps)).*handles.stimulation_data_I);
1540                    end
1541                end
1542        end
1543
1544
1545    end
1546
1547    guidata(hObject, handles);
1548
1549
1550    % --- Executes during object creation, after setting all properties.
1551    function repetition_edit_CreateFcn(hObject, eventdata, handles)
1552    % hObject    handle to repetition_edit (see GCBO)
1553    % eventdata  reserved - to be defined in a future version of MATLAB
1554    % handles    empty - handles not created until after all CreateFcns called
1555
1556    % Hint: edit controls usually have a white background on Windows.
1557    %       See ISPC and COMPUTER.
1558    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
1559        set(hObject,'BackgroundColor','white');
1560    end
1561
1562
1563    function amplitude2_edit_Callback(hObject, eventdata, handles)
1564    % hObject    handle to amplitude2_edit (see GCBO)
1565    % eventdata  reserved - to be defined in a future version of MATLAB
1566    % handles    structure with handles and user data (see GUIDATA)
1567
1568    % Hints: get(hObject,'String') returns contents of amplitude2_edit as text
1569    %        str2double(get(hObject,'String')) returns contents of amplitude2_edit as
         a double
1570
1571    % Data input by user
1572    amplitude_2_input = str2double(get(hObject,'string'));
1573
1574    if isfield(handles, 'plot_popup_value' )
1575        % handles.firing_popup_value exists - you may access it
1576    else
1577        % handles.firing_popup_value does not exist
1578        errordlg('You must choose a Waverform Type!','Invalid Input','modal')
```

```
1579        uicontrol(hObject)
1580        return
1581    end
1582
1583    if isnan(amplitude_2_input)
1584        errordlg('You must enter a numeric value','Invalid Input','modal')
1585        uicontrol(hObject)
1586        return
1587    elseif handles.amplitude_max ~=0 && amplitude_2_input > handles.amplitude_max
1588        errordlg('That value breaches the Safe Limit Range','Invalid Input','modal')
1589        uicontrol(hObject)
1590        return
1591    elseif amplitude_2_input > 5
1592        errordlg('DACs higher output is 5V','Invalid Input','modal')
1593        uicontrol(hObject)
1594        return
1595    else
1596        handles.amplitude_2 = amplitude_2_input;
1597        value = handles.plot_popup_value;
1598        string = handles.plot_popup_string;
1599        switch string{value}
1600            case 'Bi-level Lilly' % User selects bilevel Lilly
1601                amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
        amplitude_2*handles.duration_2)/handles.biphasic;
1602                handles.duration_balance_total = handles.biphasic.*(handles.
        duration_1 + handles.duration_2);
1603                handles.stimulation_fulltime = handles.duration_1+handles.
        duration_balance_total+handles.delay+handles.decay;
1604                handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
1605                handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
1606                    (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
        time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
1607                    amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
1608                    zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
1609                    -((handles.stimulation_fulltime)*handles.time_resolution)))];
1610                handles.current_data_I = handles.bi_level;
1611                handles.stimulation_data_I = handles.bi_level;
1612        end
1613
1614        handles.reps_max = 0;
1615        if handles.repetition ~= 0
1616            for n_reps = 1:handles.n_repetitions
```

```
1617            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
1618                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
1619                handles.reps_max = n_reps;
1620            end
1621            if handles.reps_max ~=0
1622                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
1623            else
1624                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
1625            end
1626        end
1627    end
1628
1629    if handles.fullstimulation_duration < handles.stimulation_fulltime
1630        set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
1631    end
1632
1633 end
1634
1635 guidata(hObject, handles);
1636
1637
1638 % --- Executes during object creation, after setting all properties.
1639 function amplitude2_edit_CreateFcn(hObject, eventdata, handles)
1640 % hObject    handle to amplitude2_edit (see GCBO)
1641 % eventdata  reserved - to be defined in a future version of MATLAB
1642 % handles    empty - handles not created until after all CreateFcns called
1643
1644 % Hint: edit controls usually have a white background on Windows.
1645 %       See ISPC and COMPUTER.
1646 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
1647    set(hObject,'BackgroundColor','white');
1648 end
1649
1650
1651
1652 function duration2_edit_Callback(hObject, eventdata, handles)
```

```
1653 % hObject    handle to duration2_edit (see GCBO)
1654 % eventdata  reserved - to be defined in a future version of MATLAB
1655 % handles    structure with handles and user data (see GUIDATA)
1656
1657 % Hints: get(hObject,'String') returns contents of duration2_edit as text
1658 %        str2double(get(hObject,'String')) returns contents of duration2_edit as
         a double
1659
1660 % Data input by user
1661 duration_2_input = str2double(get(hObject,'string'));
1662
1663 if isfield(handles, 'plot_popup_value' )
1664     % handles.firing_popup_value exists - you may access it
1665 else
1666     % handles.firing_popup_value does not exist
1667     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
1668     uicontrol(hObject)
1669     return
1670 end
1671
1672 if isnan(duration_2_input)
1673     errordlg('You must enter a numeric value','Invalid Input','modal')
1674     uicontrol(hObject)
1675     return
1676 elseif ((duration_2_input*handles.time_resolution) - floor(duration_2_input*
         handles.time_resolution)) ~= 0
1677     errordlg('Invalid input','Invalid Input','modal')
1678     uicontrol(hObject)
1679     return
1680 elseif length((0:(1/handles.time_resolution):(handles.duration_1+duration_2_input
         +(handles.biphasic*handles.duration_1)+(handles.biphasic*duration_2_input)+(
         handles.biphasic*handles.delay)+handles.delay+handles.decay))) > 930
1681     errordlg('The full duration of the pulse has to be smaller','Invalid Input','
         modal')
1682     uicontrol(hObject)
1683     return
1684 else
1685     handles.duration_2 = duration_2_input;
1686     handles.duration_balance_2 = handles.biphasic.*handles.duration_2;
1687     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
         duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
1688     handles.timescale = (0:(1/handles.time_resolution):(handles.
         stimulation_fulltime+handles.t_f));
1689     value = handles.plot_popup_value;
1690     string = handles.plot_popup_string;
1691     switch string{value}
1692         case 'Bi-level Lilly' % User selects bilevel Lilly
```

```
1693            amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
    amplitude_2*handles.duration_2)/handles.biphasic;
1694            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.duration_2);
1695            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
1696            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
1697            handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
1698                (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
    time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
1699                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
1700                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1701                -((handles.stimulation_fulltime)*handles.time_resolution)))];
1702            handles.current_data_I = handles.bi_level;
1703            handles.stimulation_data_I = handles.bi_level;
1704        end
1705
1706        handles.reps_max = 0;
1707        if handles.repetition ~= 0
1708            for n_reps = 1:handles.n_repetitions
1709                if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
1710                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
1711                    handles.reps_max = n_reps;
1712                end
1713                if handles.reps_max ~=0
1714                    handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
1715                else
1716                    handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
1717                end
1718            end
1719        end
1720
1721        if handles.fullstimulation_duration < handles.stimulation_fulltime
```

```
1722          set(handles.stim_duration_edit, 'enable', 'on','String', handles.
          stimulation_fulltime)
1723      end
1724
1725  end
1726
1727  set(handles.stim_duration_edit, 'enable', 'on','String', handles.
          stimulation_fulltime)
1728
1729  guidata(hObject, handles);
1730
1731  % --- Executes during object creation, after setting all properties.
1732  function duration2_edit_CreateFcn(hObject, eventdata, handles)
1733  % hObject    handle to duration2_edit (see GCBO)
1734  % eventdata  reserved - to be defined in a future version of MATLAB
1735  % handles    empty - handles not created until after all CreateFcns called
1736
1737  % Hint: edit controls usually have a white background on Windows.
1738  %       See ISPC and COMPUTER.
1739  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
          defaultUicontrolBackgroundColor'))
1740      set(hObject,'BackgroundColor','white');
1741  end
1742
1743
1744
1745  function decay_edit_Callback(hObject, eventdata, handles)
1746  % hObject    handle to decay_edit (see GCBO)
1747  % eventdata  reserved - to be defined in a future version of MATLAB
1748  % handles    structure with handles and user data (see GUIDATA)
1749
1750  % Hints: get(hObject,'String') returns contents of decay_edit as text
1751  %        str2double(get(hObject,'String')) returns contents of decay_edit as a
          double
1752
1753  % Data input by user
1754  decay_input = str2double(get(hObject,'string'));
1755
1756  if isfield(handles, 'plot_popup_value' )
1757      % handles.firing_popup_value exists - you may access it
1758  else
1759      % handles.firing_popup_value does not exist
1760      errordlg('You must choose a Waverform Type!','Invalid Input','modal')
1761      uicontrol(hObject)
1762      return
1763  end
1764
```

```
1765 if isnan(decay_input) || decay_input == 0 || rem(decay_input,handles.resolution)
     ~= 0
1766     errordlg('You must enter a numeric value bigger than 0 and remainder of
     division by Resolution is 0',...
1767         'Invalid Input','modal')
1768     uicontrol(hObject)
1769     return
1770 elseif length((0:(1/handles.time_resolution):(handles.duration_1+(handles.
     biphasic*(handles.duration_1+decay_input))+handles.delay+decay_input))) > 930
1771     errordlg('The full duration of the pulse has to be smaller','Invalid Input','
     modal')
1772     uicontrol(hObject)
1773     return
1774 else
1775     handles.decay = decay_input;
1776     handles.decay_steps = handles.decay/handles.resolution;
1777     handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
1778     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
     duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
1779     handles.timescale = (0:(1/handles.time_resolution):(handles.
     stimulation_fulltime+handles.t_f));
1780     value = handles.plot_popup_value;
1781     string = handles.plot_popup_string;
1782     switch string{value}
1783         case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
1784             handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
     handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
1785             i=1;
1786             total_decay_amplitude = 0;
1787             for i=1:handles.decay_steps
1788                 handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
     amplitude_1*handles.q*...
1789                     exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
     .*...
1790                     ones(1,handles.resolution*handles.time_resolution));
1791                 total_decay_amplitude = total_decay_amplitude + (handles.
     amplitude_1*handles.q*...
1792                     exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
1793             end
1794             amplitude_CB = (handles.amplitude_1*handles.duration_1 +
     total_decay_amplitude*handles.resolution)/handles.biphasic;
1795             handles.duration_balance_total = handles.biphasic.*(handles.
     duration_1 + handles.decay);
1796             handles.stimulation_fulltime = handles.duration_1+handles.
     duration_balance_total+handles.delay+handles.decay;
1797             handles.timescale = (0:(1/handles.time_resolution):(handles.
     stimulation_fulltime+handles.t_f));
```

```
1798            handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
1799                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
1800                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1801                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1802            handles.current_data_I = handles.quasi_trap;
1803            handles.stimulation_data_I = handles.quasi_trap;
1804        case 'Tri-Exponential' % User selects tri-exponential
1805            handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
1806            j_i = 1;
1807            j_d = 1;
1808            total_increase_amplitude = 0;
1809            total_decay_amplitude = 0;
1810            for j_i=1:handles.increase_steps
1811                handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
    resolution*handles.time_resolution)...
1812                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
1813                    ones(1,handles.resolution*handles.time_resolution));
1814                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
1815                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
1816            end
1817            for j_d=1:handles.decay_steps
1818                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
1819                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
1820                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
1821                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
1822                    ones(1,handles.resolution*handles.time_resolution));
1823                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
1824                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
1825                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
1826                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
1827            end
```

```
1828            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
1829            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
1830            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
1831            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
1832            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
1833                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
1834                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
1835                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1836            handles.current_data_I = handles.tri_exp;
1837            handles.stimulation_data_I = handles.tri_exp;
1838        end
1839
1840        handles.reps_max = 0;
1841        if handles.repetition ~= 0
1842            for n_reps = 1:handles.n_repetitions
1843                if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
1844                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
1845                   handles.reps_max = n_reps;
1846                end
1847                if handles.reps_max ~=0
1848                    handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
1849                else
1850                    handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
1851                end
1852            end
1853        end
1854
1855        if handles.fullstimulation_duration < handles.stimulation_fulltime
1856            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
1857        end
```

```
1858
1859 end
1860
1861 set(handles.stim_duration_edit, 'enable', 'on','String', handles.
         stimulation_fulltime)
1862
1863 guidata(hObject, handles);
1864
1865 % --- Executes during object creation, after setting all properties.
1866 function decay_edit_CreateFcn(hObject, eventdata, handles)
1867 % hObject    handle to decay_edit (see GCBO)
1868 % eventdata  reserved - to be defined in a future version of MATLAB
1869 % handles    empty - handles not created until after all CreateFcns called
1870
1871 % Hint: edit controls usually have a white background on Windows.
1872 %       See ISPC and COMPUTER.
1873 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
         defaultUicontrolBackgroundColor'))
1874     set(hObject,'BackgroundColor','white');
1875 end
1876
1877
1878
1879 function resolution_edit_Callback(hObject, eventdata, handles)
1880 % hObject    handle to resolution_edit (see GCBO)
1881 % eventdata  reserved - to be defined in a future version of MATLAB
1882 % handles    structure with handles and user data (see GUIDATA)
1883
1884 % Hints: get(hObject,'String') returns contents of resolution_edit as text
1885 %        str2double(get(hObject,'String')) returns contents of resolution_edit as
          a double
1886
1887 % Data input by user
1888 resolution_input = str2double(get(hObject,'string'));
1889
1890 if isfield(handles, 'plot_popup_value' )
1891     % handles.firing_popup_value exists - you may access it
1892 else
1893     % handles.firing_popup_value does not exist
1894     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
1895     uicontrol(hObject)
1896     return
1897 end
1898
1899 if isnan(resolution_input) || resolution_input == 0 || rem(handles.decay,
         resolution_input) ~= 0 ||...
1900         ((resolution_input*handles.time_resolution) > 0 && (resolution_input*
         handles.time_resolution) < 1)
```

```
1901      errordlg('You must enter a numeric value between ]0,1], whos remainder of
          dividing Decay is 0 and product by Time Resolution (40 by defenition) is an
          integer',...
1902          'Invalid Input','modal')
1903      uicontrol(hObject)
1904      return
1905  else
1906      handles.resolution = resolution_input;
1907      handles.decay_steps = handles.decay/handles.resolution;
1908      handles.burst_steps = handles.duration_1/handles.resolution;
1909      handles.increase_steps = handles.duration_1/handles.resolution;
1910      handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
1911      handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
          duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
1912      handles.timescale = (0:(1/handles.time_resolution):(handles.
          stimulation_fulltime+handles.t_f));
1913      value = handles.plot_popup_value;
1914      string = handles.plot_popup_string;
1915      switch string{value}
1916          case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
1917              handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
          handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
1918              i=1;
1919              total_decay_amplitude = 0;
1920              for i=1:handles.decay_steps
1921                  handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
          amplitude_1*handles.q*...
1922                      exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
          .*...
1923                      ones(1,handles.resolution*handles.time_resolution));
1924                  total_decay_amplitude = total_decay_amplitude + (handles.
          amplitude_1*handles.q*...
1925                      exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
1926              end
1927              amplitude_CB = (handles.amplitude_1*handles.duration_1 +
          total_decay_amplitude*handles.resolution)/handles.biphasic;
1928              handles.duration_balance_total = handles.biphasic.*(handles.
          duration_1 + handles.decay);
1929              handles.stimulation_fulltime = handles.duration_1+handles.
          duration_balance_total+handles.delay+handles.decay;
1930              handles.timescale = (0:(1/handles.time_resolution):(handles.
          stimulation_fulltime+handles.t_f));
1931              handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
          delay*handles.time_resolution))...
1932                  amplitude_CB.*ones(1,handles.duration_balance_total*handles.
          time_resolution)...
1933                  zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
          time_resolution)...
```

```
1934                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1935            handles.current_data_I = handles.quasi_trap;
1936            handles.stimulation_data_I = handles.quasi_trap;
1937        case 'Tri-Exponential' % User selects tri-exponential
1938            handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
1939            j_i = 1;
1940            j_d = 1;
1941            total_increase_amplitude = 0;
1942            total_decay_amplitude = 0;
1943            for j_i=1:handles.increase_steps
1944                handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
    resolution*handles.time_resolution)...
1945                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
1946                    ones(1,handles.resolution*handles.time_resolution));
1947                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
1948                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
1949            end
1950            for j_d=1:handles.decay_steps
1951                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
1952                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
1953                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
1954                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
1955                    ones(1,handles.resolution*handles.time_resolution));
1956                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
1957                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
1958                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
1959                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
1960            end
1961            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
1962            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
1963            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
```

```
1964            handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
1965            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
        handles.time_resolution))...
1966                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
1967                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
1968                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
1969            handles.current_data_I = handles.tri_exp;
1970            handles.stimulation_data_I = handles.tri_exp;
1971        case 'Bursting' % User selects bursting strategies
1972            if (((resolution_input/2)*handles.time_resolution) > 0 && ((
        resolution_input/2)*handles.time_resolution) < 1)
1973                errordlg('You must enter a numeric value whos division by 2
        followed of multiplication by Time Resolution (100 by defenition) is an
        integer',...
1974                    'Invalid Input','modal')
1975                uicontrol(hObject)
1976                return
1977            else
1978                handles.duration_balance_1 = handles.biphasic.*(handles.
        duration_1/2);
1979                handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
1980                k=1;
1981                for k=1:handles.burst_steps
1982                    handles.bursting = horzcat(handles.bursting,[-handles.
        amplitude_1.*...
1983                        ones(1,(handles.resolution/2)*handles.time_resolution)
        ...
1984                        zeros(1,(handles.resolution/2)*handles.time_resolution)])
        ;
1985                end
1986                handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.
        delay*handles.time_resolution))...
1987                    ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*
        ones(1,handles.duration_balance_1*handles.time_resolution)...
1988                    zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
1989                    -((handles.stimulation_fulltime)*handles.time_resolution)))])
        ;
1990                handles.current_data_I = handles.bursting;
1991                handles.stimulation_data_I = handles.bursting;
1992            end
1993    end
1994
1995    handles.reps_max = 0;
1996    if handles.repetition ~= 0
```

```
1997          for n_reps = 1:handles.n_repetitions
1998              if handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
      amplitude_1*((handles.rep_amp_ratio).^(n_reps)) || handles.amplitude_max <=
      handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps)) ) && handles.reps_max
       == 0
1999                      handles.reps_max = n_reps;
2000              end
2001              if handles.reps_max ~=0
2002                  handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
      handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
      .^(handles.reps_max)).*handles.stimulation_data_I);
2003              else
2004                  handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
      handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
      .^(n_reps)).*handles.stimulation_data_I);
2005              end
2006          end
2007      end
2008
2009      if handles.fullstimulation_duration < handles.stimulation_fulltime
2010          set(handles.stim_duration_edit, 'enable', 'on','String', handles.
      stimulation_fulltime)
2011      end
2012
2013  end
2014
2015  guidata(hObject, handles);
2016
2017  % --- Executes during object creation, after setting all properties.
2018  function resolution_edit_CreateFcn(hObject, eventdata, handles)
2019  % hObject    handle to resolution_edit (see GCBO)
2020  % eventdata  reserved - to be defined in a future version of MATLAB
2021  % handles    empty - handles not created until after all CreateFcns called
2022
2023  % Hint: edit controls usually have a white background on Windows.
2024  %       See ISPC and COMPUTER.
2025  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
      defaultUicontrolBackgroundColor'))
2026      set(hObject,'BackgroundColor','white');
2027  end
2028
2029  % --- Executes on button press in plot_pushbutton.
2030  function plot_pushbutton_Callback(hObject, eventdata, handles)
2031  % hObject    handle to plot_pushbutton (see GCBO)
2032  % eventdata  reserved - to be defined in a future version of MATLAB
2033  % handles    structure with handles and user data (see GUIDATA)
2034
2035  % Display surf plot of the currently selected data
```

```
2036 handles.timescale = (0:(1/handles.time_resolution):(length(handles.current_data_I
         )-1)/handles.time_resolution);
2037 % handles.timescale = (0:(1/handles.time_resolution):(handles.
         stimulation_fulltime+handles.t_f));
2038 axes(handles.plot_axes);
2039 plot(handles.timescale,handles.current_data_I);
2040 if (min(handles.current_data_I)-0.1) > -(min(handles.current_data_I)-0.1)
2041     ylim([(min(handles.current_data_I)-0.1),(max(handles.current_data_I)-0.1)]);
2042 else
2043     ylim([(min(handles.current_data_I)-0.1),-(min(handles.current_data_I)-0.1)]);
2044 end
2045 xlim([0,((length(handles.current_data_I)-1)/handles.time_resolution)]);
2046 %xlim([0,((length(handles.current_data_I)-1)/handles.time_resolution)]);
2047 ylabel('Amplitude (V)');
2048 xlabel('Time (ms)');
2049
2050 length(handles.stimulation_data_I);
2051
2052 if length(handles.stimulation_data_I) > 930
2053     errordlg('An unknown error occurred. Probably related to length of pulse.
         Values in boxes were reseted. If problem pressists, change waveform once',...
2054         'Unknown Error','modal')
2055     switch handles.plot_popup_string{handles.plot_popup_value}
2056         case 'Rectangular' % User selects pulse wave
2057             % Define non interest variables to the original value
2058             handles.amplitude_1 = 1;
2059             handles.duration_1 = 1;
2060             handles.amplitude_2 = 0;
2061             handles.duration_2 = 0;
2062             handles.biphasic = 10;
2063             handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
2064             handles.duration_balance_2 = 0;
2065             handles.decay = 0;
2066             handles.delay = 1;
2067             handles.resolution = 0.25;
2068             handles.repetition = 0;
2069             handles.rep_amp_ratio = 1;
2070             handles.stimulation_fulltime = handles.duration_1+handles.duration_2+
         handles.duration_balance_1+handles.duration_balance_2+handles.delay+handles.
         decay;
2071             handles.fullstimulation_duration = handles.stimulation_fulltime;
2072             handles.timescale = (0:(1/handles.time_resolution):(handles.
         stimulation_fulltime+handles.t_f));
2073             %Building the pulse wave
2074             handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-
         handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
2075                 zeros(1,(handles.delay*handles.time_resolution))...
```

```
2076                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
2077                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2078                -((handles.stimulation_fulltime)*handles.time_resolution))];
2079            %
2080            handles.current_data_I = handles.pulse;
2081            handles.stimulation_data_I = handles.pulse;
2082            % Define with edit fields should work and define its initial values
2083            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
2084            set(handles.duration1_edit, 'enable', 'on','String', handles.
    duration_1)
2085            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2086            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2087            set(handles.amplitude2_edit, 'enable', 'off')
2088            set(handles.duration2_edit, 'enable', 'off')
2089            set(handles.decay_edit, 'enable', 'off')
2090            set(handles.resolution_edit, 'enable', 'off')
2091            set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
2092            set(handles.Q_exp_edit, 'enable', 'off')
2093            set(handles.Q2_exp_edit, 'enable', 'off')
2094            set(handles.A_exp_edit, 'enable', 'off')
2095            set(handles.A2_exp_edit, 'enable', 'off')
2096            set(handles.B_exp_edit, 'enable', 'off')
2097            set(handles.B2_exp_edit, 'enable', 'off')
2098            set(handles.C_exp_edit, 'enable', 'off')
2099            set(handles.C2_exp_edit, 'enable', 'off')
2100            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
2101            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
2102
2103        case 'Bi-level Lilly' % User selects bilevel Lilly
2104            handles.amplitude_1 = 1;
2105            handles.duration_1 = 1;
2106            handles.amplitude_2 = 0.5;
2107            handles.duration_2 = 1;
2108            handles.biphasic = 10;
2109            handles.delay = 1;
2110            handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
2111            handles.duration_balance_2 = handles.biphasic.*handles.duration_2;
2112            handles.decay = 0;
2113            handles.resolution = 0.25;
2114            handles.repetition = 0;
2115            handles.rep_amp_ratio = 1;
2116            % Building the Bilevel Lilly
```

```
2117            amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
        amplitude_2*handles.duration_2)/handles.biphasic;
2118            handles.duration_balance_total = handles.biphasic.*(handles.
        duration_1 + handles.duration_2);
2119            handles.stimulation_fulltime = handles.duration_1+handles.
        duration_balance_total+handles.delay+handles.decay;
2120            handles.fullstimulation_duration = handles.stimulation_fulltime;
2121            handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
2122            handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
2123                (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
        time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
2124                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
2125                zeros(1,(length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
2126                -((handles.stimulation_fulltime)*handles.time_resolution))];
2127            %
2128            handles.current_data_I = handles.bi_level;
2129            handles.stimulation_data_I = handles.bi_level;
2130            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
        amplitude_1)
2131            set(handles.duration1_edit, 'enable', 'on','String', handles.
        duration_1)
2132            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2133            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2134            set(handles.amplitude2_edit, 'enable', 'on','String', handles.
        amplitude_2)
2135            set(handles.duration2_edit, 'enable', 'on','String', handles.
        duration_2 )
2136            set(handles.decay_edit, 'enable', 'off')
2137            set(handles.resolution_edit, 'enable', 'off')
2138            set(handles.repetition_edit, 'enable', 'on','String', handles.
        repetition)
2139            set(handles.Q_exp_edit, 'enable', 'off')
2140            set(handles.Q2_exp_edit, 'enable', 'off')
2141            set(handles.A_exp_edit, 'enable', 'off')
2142            set(handles.A2_exp_edit, 'enable', 'off')
2143            set(handles.B_exp_edit, 'enable', 'off')
2144            set(handles.B2_exp_edit, 'enable', 'off')
2145            set(handles.C_exp_edit, 'enable', 'off')
2146            set(handles.C2_exp_edit, 'enable', 'off')
2147            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
2148            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
        rep_amp_ratio)
2149
```

```
2150        case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
2151            handles.amplitude_1 = 1;
2152            handles.duration_1 = 1;
2153            handles.amplitude_2 = 0.5;
2154            handles.duration_2 = 1;
2155            handles.biphasic = 10;
2156            handles.delay = 1;
2157            handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
2158            handles.duration_balance_2 = 0;
2159            handles.decay = 1;
2160            handles.resolution = 0.25;
2161            handles.repetition = 0;
2162            handles.q = 1;
2163            handles.q2 = 2.1972e+03;
2164            handles.rep_amp_ratio = 1;
2165            % Building the Quasi-Trapezoidal Wave
2166            handles.decay_steps = handles.decay/handles.resolution;
2167            handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
2168            i=1;
2169            total_decay_amplitude = 0;
2170            for i=1:handles.decay_steps
2171                handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
    amplitude_1*handles.q*...
2172                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
    .*...
2173                    ones(1,handles.resolution*handles.time_resolution));
2174                total_decay_amplitude = total_decay_amplitude + (handles.
    amplitude_1*handles.q*...
2175                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
2176            end
2177            amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
2178            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
2179            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
2180            handles.fullstimulation_duration = handles.stimulation_fulltime;
2181            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2182            handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
2183                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
2184                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2185                -((handles.stimulation_fulltime)*handles.time_resolution))]);
2186            %
```

```
2187            handles.current_data_I = handles.quasi_trap;
2188            handles.stimulation_data_I = handles.quasi_trap;
2189            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
        amplitude_1)
2190            set(handles.duration1_edit, 'enable', 'on','String', handles.
        duration_1)
2191            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2192            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2193            set(handles.amplitude2_edit, 'enable', 'off')
2194            set(handles.duration2_edit, 'enable', 'off')
2195            set(handles.decay_edit, 'enable', 'on','String', handles.decay)
2196            set(handles.resolution_edit, 'enable', 'on','String', handles.
        resolution)
2197            set(handles.repetition_edit, 'enable', 'on','String', handles.
        repetition)
2198            set(handles.Q_exp_edit, 'enable', 'on','String', handles.q)
2199            set(handles.Q2_exp_edit, 'enable', 'on', 'String', handles.q2)
2200            set(handles.A_exp_edit, 'enable', 'off')
2201            set(handles.A2_exp_edit, 'enable', 'off')
2202            set(handles.B_exp_edit, 'enable', 'off')
2203            set(handles.B2_exp_edit, 'enable', 'off')
2204            set(handles.C_exp_edit, 'enable', 'off')
2205            set(handles.C2_exp_edit, 'enable', 'off')
2206            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
2207            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
        rep_amp_ratio)
2208
2209        case 'Tri-Exponential' % User selects tri-exponential
2210            handles.amplitude_1 = 1;
2211            handles.duration_1 = 1;
2212            handles.amplitude_2 = 0.5;
2213            handles.duration_2 = 1;
2214            handles.biphasic = 10;
2215            handles.delay = 1;
2216            handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
2217            handles.duration_balance_2 = 0;
2218            handles.decay = 1;
2219            handles.repetition = 0;
2220            handles.resolution = 0.25;
2221            handles.a = 0.2;
2222            handles.a2 = 1.197e+03;
2223            handles.b = 0.3;
2224            handles.b2 = 2.197e+03;
2225            handles.c = 0.5;
2226            handles.c2 = 3.197e+03;
2227            handles.repetition = 0;
2228            handles.rep_amp_ratio = 1;
```

```
2229            % Building the Tri-Exponential Wave
2230            handles.increase_steps = handles.duration_1/handles.resolution;
2231            handles.decay_steps = handles.decay/handles.resolution;
2232            handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
2233            j_i = 1;
2234            j_d = 1;
2235            total_increase_amplitude = 0;
2236            total_decay_amplitude = 0;
2237            for j_i=1:handles.increase_steps
2238                handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
    resolution*handles.time_resolution)...
2239                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
2240                    ones(1,handles.resolution*handles.time_resolution));
2241                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
2242                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
2243            end
2244            for j_d=1:handles.decay_steps
2245                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
2246                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
2247                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
2248                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
2249                    ones(1,handles.resolution*handles.time_resolution));
2250                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
2251                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
2252                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
2253                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
2254            end
2255            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
2256            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
2257            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
2258            handles.fullstimulation_duration = handles.stimulation_fulltime;
```

```
2259            handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
2260            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
        handles.time_resolution))...
2261                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
2262                zeros(1,(length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
2263                -((handles.stimulation_fulltime)*handles.time_resolution))]);
2264            %
2265            handles.current_data_I = handles.tri_exp;
2266            handles.stimulation_data_I = handles.tri_exp;
2267            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
        amplitude_1)
2268            set(handles.duration1_edit, 'enable', 'on','String', handles.
        duration_1)
2269            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2270            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2271            set(handles.amplitude2_edit, 'enable', 'off')
2272            set(handles.duration2_edit, 'enable', 'off')
2273            set(handles.decay_edit, 'enable', 'on','String', handles.decay)
2274            set(handles.resolution_edit, 'enable', 'on','String', handles.
        resolution)
2275            set(handles.repetition_edit, 'enable', 'on','String', handles.
        repetition)
2276            set(handles.Q_exp_edit, 'enable', 'off')
2277            set(handles.Q2_exp_edit, 'enable', 'off')
2278            set(handles.A_exp_edit, 'enable', 'on','String', handles.a)
2279            set(handles.A2_exp_edit, 'enable', 'on', 'String', handles.a2)
2280            set(handles.B_exp_edit, 'enable', 'on','String', handles.b)
2281            set(handles.B2_exp_edit, 'enable', 'on', 'String', handles.b2)
2282            set(handles.C_exp_edit, 'enable', 'on','String', handles.c)
2283            set(handles.C2_exp_edit, 'enable', 'on', 'String', handles.c2)
2284            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
2285            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
        rep_amp_ratio)
2286
2287        case 'Bursting' % User selects bursting strategies
2288            handles.amplitude_1 = 1;
2289            handles.duration_1 = 1;
2290            handles.amplitude_2 = 0.5;
2291            handles.duration_2 = 1;
2292            handles.biphasic = 10;
2293            handles.delay = 1;
2294            handles.duration_balance_2 = 0;
2295            handles.decay = 0;
```

```
2296            handles.duration_balance_1 = handles.biphasic.*(handles.duration_1/2)
    ;
2297            handles.resolution = 0.25;
2298            handles.repetition = 0;
2299            handles.rep_amp_ratio = 1;
2300            handles.stimulation_fulltime = handles.duration_1+handles.duration_2+
    handles.duration_balance_1+handles.duration_balance_2+handles.delay+handles.
    decay;
2301            handles.fullstimulation_duration = handles.stimulation_fulltime;
2302            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2303            % Building the Bursting Wave
2304            handles.burst_steps = handles.duration_1/handles.resolution;
2305            handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
2306            k=1;
2307            for k=1:handles.burst_steps
2308                handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1
    .*...
2309                    ones(1,(handles.resolution/2)*handles.time_resolution) ...
2310                    zeros(1,(handles.resolution/2)*handles.time_resolution)]);
2311            end
2312            handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*
    handles.time_resolution))...
2313                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
2314                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2315                -((handles.stimulation_fulltime)*handles.time_resolution))]);
2316            %
2317            handles.current_data_I = handles.bursting;
2318            handles.stimulation_data_I = handles.bursting;
2319            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
2320            set(handles.duration1_edit, 'enable', 'on','String', handles.
    duration_1)
2321            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2322            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2323            set(handles.amplitude2_edit, 'enable', 'off')
2324            set(handles.duration2_edit, 'enable', 'off')
2325            set(handles.decay_edit, 'enable', 'off')
2326            set(handles.resolution_edit, 'enable', 'on','String', handles.
    resolution)
2327            set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
2328            set(handles.Q_exp_edit, 'enable', 'off')
2329            set(handles.Q2_exp_edit, 'enable', 'off')
2330            set(handles.A_exp_edit, 'enable', 'off')
2331            set(handles.A2_exp_edit, 'enable', 'off')
```

```
2332                set(handles.B_exp_edit, 'enable', 'off')
2333                set(handles.B2_exp_edit, 'enable', 'off')
2334                set(handles.C_exp_edit, 'enable', 'off')
2335                set(handles.C2_exp_edit, 'enable', 'off')
2336                set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
2337                set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
        rep_amp_ratio)
2338                uicontrol(hObject)
2339                return
2340            end
2341        end
2342
2343
2344
2345
2346
2347
2348
2349        % --- Executes on button press in upload_pushbutton.
2350        function upload_pushbutton_Callback(hObject, eventdata, handles)
2351        % hObject    handle to upload_pushbutton (see GCBO)
2352        % eventdata  reserved - to be defined in a future version of MATLAB
2353        % handles    structure with handles and user data (see GUIDATA)
2354
2355        if isfield(handles, 'plot_popup_value' )
2356            % handles.plot_popup_value exists - you may access it
2357        else
2358            % handles.plot_popup_value does not exist returns to interface without
2359            % any action
2360            errordlg('You must choose a Waverform Type!','Invalid Input','modal')
2361            uicontrol(hObject)
2362            return
2363        end
2364
2365        if isfield(handles, 'firing_popup_value' )
2366            % handles.firing_popup_value exists - you may access it
2367        else
2368            % handles.firing_popup_value does not exist returns to interface without
2369            % any action
2370            errordlg('You must choose a Firing Type!','Invalid Input','modal')
2371            uicontrol(hObject)
2372            return
2373        end
2374
2375        if isfield(handles, 'channel_popup_value' )
2376            % handles.channel_popup_value exists - you may access it
2377        else
```

```
2378      % handles.channel_popup_value does not exist returns to interface without
2379      % any action
2380      errordlg('You must choose a Channel!','Invalid Input','modal')
2381      uicontrol(hObject)
2382      return
2383  end
2384
2385  if isfield(handles, 'COM_popup_value' )
2386      % handles.COM_popup_value exists - you may access it
2387  else
2388      % handles.COM_popup_value does not exist
2389      errordlg('No COM port selected!','Invalid Input','modal')
2390      % searches for new COM ports
2391      serialInfo = instrhwinfo('serial');
2392      % if there are no ports returns to interface without
2393      % any action
2394      if isempty(serialInfo.SerialPorts) == 1
2395          errordlg('There are no available COM ports','Invalid Input','modal')
2396          uicontrol(hObject)
2397          return
2398      else % if there is ports, makes them available to user choice and also
2399          % returns to interface with no action
2400          n_coms = 1;
2401          for n_coms = 1:length(serialInfo.SerialPorts)
2402              string_list(n_coms+1) = serialInfo.SerialPorts(n_coms);
2403          end
2404          set(handles.COM_popup,'String',string_list);
2405      end
2406
2407      uicontrol(hObject)
2408      return
2409  end
2410
2411  % if, for some reason, the control of length as failed throughout the
2412  % interface, this is a last control point. If is bigger than the maximum
2413  % length allowed, resets all options
2414  if length(handles.stimulation_data_I) > 930
2415      errordlg('An unknown error occurred. Values in boxes were reseted.','Unknown
         Error','modal')
2416      switch handles.plot_popup_string{handles.plot_popup_value}
2417          case 'Rectangular' % User selects pulse wave
2418              % Define non interest variables to the original value
2419              handles.amplitude_1 = 1;
2420              handles.duration_1 = 1;
2421              handles.amplitude_2 = 0;
2422              handles.duration_2 = 0;
2423              handles.biphasic = 10;
2424              handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
```

```
2425            handles.duration_balance_2 = 0;
2426            handles.decay = 0;
2427            handles.delay = 1;
2428            handles.resolution = 0.25;
2429            handles.repetition = 0;
2430            handles.rep_amp_ratio = 1;
2431            handles.stimulation_fulltime = handles.duration_1+handles.duration_2+
    handles.duration_balance_1+handles.duration_balance_2+handles.delay+handles.
    decay;
2432            handles.fullstimulation_duration = handles.stimulation_fulltime;
2433            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2434            %Building the pulse wave
2435            handles.pulse = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
2436                zeros(1,(handles.delay*handles.time_resolution))...
2437                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
2438                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2439                -((handles.stimulation_fulltime)*handles.time_resolution))];
2440            %
2441            handles.current_data_I = handles.pulse;
2442            handles.stimulation_data_I = handles.pulse;
2443            % Define with edit fields should work and define its initial values
2444            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
2445            set(handles.duration1_edit, 'enable', 'on','String', handles.
    duration_1)
2446            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2447            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2448            set(handles.amplitude2_edit, 'enable', 'off')
2449            set(handles.duration2_edit, 'enable', 'off')
2450            set(handles.decay_edit, 'enable', 'off')
2451            set(handles.resolution_edit, 'enable', 'off')
2452            set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
2453            set(handles.Q_exp_edit, 'enable', 'off')
2454            set(handles.Q2_exp_edit, 'enable', 'off')
2455            set(handles.A_exp_edit, 'enable', 'off')
2456            set(handles.A2_exp_edit, 'enable', 'off')
2457            set(handles.B_exp_edit, 'enable', 'off')
2458            set(handles.B2_exp_edit, 'enable', 'off')
2459            set(handles.C_exp_edit, 'enable', 'off')
2460            set(handles.C2_exp_edit, 'enable', 'off')
2461            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
```

```
2462            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
2463
2464        case 'Bi-level Lilly' % User selects bilevel Lilly
2465            handles.amplitude_1 = 1;
2466            handles.duration_1 = 1;
2467            handles.amplitude_2 = 0.5;
2468            handles.duration_2 = 1;
2469            handles.biphasic = 10;
2470            handles.delay = 1;
2471            handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
2472            handles.duration_balance_2 = handles.biphasic.*handles.duration_2;
2473            handles.decay = 0;
2474            handles.resolution = 0.25;
2475            handles.repetition = 0;
2476            handles.rep_amp_ratio = 1;
2477            % Building the Bilevel Lilly
2478            amplitude_CB = (handles.amplitude_1*handles.duration_1 + handles.
    amplitude_2*handles.duration_2)/handles.biphasic;
2479            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.duration_2);
2480            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
2481            handles.fullstimulation_duration = handles.stimulation_fulltime;
2482            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2483            handles.bi_level = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)...
2484                (-handles.amplitude_2).*ones(1,handles.duration_2*handles.
    time_resolution) zeros(1,(handles.delay*handles.time_resolution))...
2485                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
2486                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2487                -((handles.stimulation_fulltime)*handles.time_resolution))];
2488            %
2489            handles.current_data_I = handles.bi_level;
2490            handles.stimulation_data_I = handles.bi_level;
2491            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
2492            set(handles.duration1_edit, 'enable', 'on','String', handles.
    duration_1)
2493            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2494            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2495            set(handles.amplitude2_edit, 'enable', 'on','String', handles.
    amplitude_2)
2496            set(handles.duration2_edit, 'enable', 'on','String', handles.
    duration_2 )
```

```
2497            set(handles.decay_edit, 'enable', 'off')
2498            set(handles.resolution_edit, 'enable', 'off')
2499            set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
2500            set(handles.Q_exp_edit, 'enable', 'off')
2501            set(handles.Q2_exp_edit, 'enable', 'off')
2502            set(handles.A_exp_edit, 'enable', 'off')
2503            set(handles.A2_exp_edit, 'enable', 'off')
2504            set(handles.B_exp_edit, 'enable', 'off')
2505            set(handles.B2_exp_edit, 'enable', 'off')
2506            set(handles.C_exp_edit, 'enable', 'off')
2507            set(handles.C2_exp_edit, 'enable', 'off')
2508            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
2509            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
2510
2511        case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
2512            handles.amplitude_1 = 1;
2513            handles.duration_1 = 1;
2514            handles.amplitude_2 = 0.5;
2515            handles.duration_2 = 1;
2516            handles.biphasic = 10;
2517            handles.delay = 1;
2518            handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
2519            handles.duration_balance_2 = 0;
2520            handles.decay = 1;
2521            handles.repetition = 0;
2522            handles.q = 1;
2523            handles.q2 = 2.1972e+03;
2524            handles.rep_amp_ratio = 1;
2525            % Building the Quasi-Trapezoidal Wave
2526            handles.decay_steps = handles.decay/handles.resolution;
2527            handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
    handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
2528            i=1;
2529            total_decay_amplitude = 0;
2530            for i=1:handles.decay_steps
2531                handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
    amplitude_1*handles.q*...
2532                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
    .*...
2533                    ones(1,handles.resolution*handles.time_resolution));
2534                total_decay_amplitude = total_decay_amplitude + (handles.
    amplitude_1*handles.q*...
2535                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
2536            end
```

```
2537            amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
2538            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
2539            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
2540            handles.fullstimulation_duration = handles.stimulation_fulltime;
2541            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2542            handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
2543                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
2544                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2545                -((handles.stimulation_fulltime)*handles.time_resolution))]);
2546            %
2547            handles.current_data_I = handles.quasi_trap;
2548            handles.stimulation_data_I = handles.quasi_trap;
2549            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
2550            set(handles.duration1_edit, 'enable', 'on','String', handles.
    duration_1)
2551            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2552            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2553            set(handles.amplitude2_edit, 'enable', 'off')
2554            set(handles.duration2_edit, 'enable', 'off')
2555            set(handles.decay_edit, 'enable', 'on','String', handles.decay)
2556            set(handles.resolution_edit, 'enable', 'on','String', handles.
    resolution)
2557            set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
2558            set(handles.Q_exp_edit, 'enable', 'on','String', handles.q)
2559            set(handles.Q2_exp_edit, 'enable', 'on', 'String', handles.q2)
2560            set(handles.A_exp_edit, 'enable', 'off')
2561            set(handles.A2_exp_edit, 'enable', 'off')
2562            set(handles.B_exp_edit, 'enable', 'off')
2563            set(handles.B2_exp_edit, 'enable', 'off')
2564            set(handles.C_exp_edit, 'enable', 'off')
2565            set(handles.C2_exp_edit, 'enable', 'off')
2566            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
2567            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
2568
2569        case 'Tri-Exponential' % User selects tri-exponential
2570            handles.amplitude_1 = 1;
2571            handles.duration_1 = 1;
```

```
2572            handles.amplitude_2 = 0.5;
2573            handles.duration_2 = 1;
2574            handles.biphasic = 10;
2575            handles.delay = 1;
2576            handles.duration_balance_1 = handles.biphasic.*handles.duration_1;
2577            handles.duration_balance_2 = 0;
2578            handles.decay = 1;
2579            handles.repetition = 0;
2580            handles.resolution = 0.25;
2581            handles.a = 0.2;
2582            handles.a2 = 1.197e+03;
2583            handles.b = 0.3;
2584            handles.b2 = 2.197e+03;
2585            handles.c = 0.5;
2586            handles.c2 = 3.197e+03;
2587            handles.repetition = 0;
2588            handles.rep_amp_ratio = 1;
2589            % Building the Tri-Exponential Wave
2590            handles.increase_steps = handles.duration_1/handles.resolution;
2591            handles.decay_steps = handles.decay/handles.resolution;
2592            handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
2593            j_i = 1;
2594            j_d = 1;
2595            total_increase_amplitude = 0;
2596            total_decay_amplitude = 0;
2597            for j_i=1:handles.increase_steps
2598                handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
resolution*handles.time_resolution)...
2599                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
handles.t_i)-handles.t_i)))/handles.time_resolution).*...
2600                    ones(1,handles.resolution*handles.time_resolution));
2601                total_increase_amplitude = total_increase_amplitude + (((handles.
amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
resolution*handles.time_resolution)...
2602                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
handles.t_i)-handles.t_i))/handles.time_resolution);
2603            end
2604            for j_d=1:handles.decay_steps
2605                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
2606                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
+(handles.b*...
2607                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
+(handles.c*...
2608                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
).*...
2609                    ones(1,handles.resolution*handles.time_resolution));
```

```
2610                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
2611                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
2612                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
2613                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
2614            end
2615            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
2616            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
2617            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
2618            handles.fullstimulation_duration = handles.stimulation_fulltime;
2619            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2620            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
2621                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
2622                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2623                -((handles.stimulation_fulltime)*handles.time_resolution))]);
2624            %
2625            handles.current_data_I = handles.tri_exp;
2626            handles.stimulation_data_I = handles.tri_exp;
2627            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
2628            set(handles.duration1_edit, 'enable', 'on','String', handles.
    duration_1)
2629            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2630            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2631            set(handles.amplitude2_edit, 'enable', 'off')
2632            set(handles.duration2_edit, 'enable', 'off')
2633            set(handles.decay_edit, 'enable', 'on','String', handles.decay)
2634            set(handles.resolution_edit, 'enable', 'on','String', handles.
    resolution)
2635            set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
2636            set(handles.Q_exp_edit, 'enable', 'off')
2637            set(handles.Q2_exp_edit, 'enable', 'off')
2638            set(handles.A_exp_edit, 'enable', 'on','String', handles.a)
2639            set(handles.A2_exp_edit, 'enable', 'on', 'String', handles.a2)
2640            set(handles.B_exp_edit, 'enable', 'on','String', handles.b)
2641            set(handles.B2_exp_edit, 'enable', 'on', 'String', handles.b2)
2642            set(handles.C_exp_edit, 'enable', 'on','String', handles.c)
```

```
2643            set(handles.C2_exp_edit, 'enable', 'on', 'String', handles.c2)
2644            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
2645            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
2646
2647        case 'Bursting' % User selects bursting strategies
2648            handles.amplitude_1 = 1;
2649            handles.duration_1 = 1;
2650            handles.amplitude_2 = 0.5;
2651            handles.duration_2 = 1;
2652            handles.biphasic = 10;
2653            handles.delay = 1;
2654            handles.duration_balance_2 = 0;
2655            handles.decay = 0;
2656            handles.duration_balance_1 = handles.biphasic.*(handles.duration_1/2)
    ;
2657            handles.resolution = 0.25;
2658            handles.repetition = 0;
2659            handles.rep_amp_ratio = 1;
2660            handles.stimulation_fulltime = handles.duration_1+handles.duration_2+
    handles.duration_balance_1+handles.duration_balance_2+handles.delay+handles.
    decay;
2661            handles.fullstimulation_duration = handles.stimulation_fulltime;
2662            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2663            % Building the Bursting Wave
2664            handles.burst_steps = handles.duration_1/handles.resolution;
2665            handles.bursting = zeros(1,handles.t_i*handles.time_resolution);
2666            k=1;
2667            for k=1:handles.burst_steps
2668                handles.bursting = horzcat(handles.bursting,[-handles.amplitude_1
    .*...
2669                    ones(1,(handles.resolution/2)*handles.time_resolution) ...
2670                    zeros(1,(handles.resolution/2)*handles.time_resolution)]);
2671            end
2672            handles.bursting = horzcat(handles.bursting, [zeros(1,(handles.delay*
    handles.time_resolution))...
2673                ((handles.amplitude_1*handles.duration_1)/handles.biphasic).*ones
    (1,handles.duration_balance_1*handles.time_resolution)...
2674                zeros(1,(length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2675                -((handles.stimulation_fulltime)*handles.time_resolution))]);
2676            %
2677            handles.current_data_I = handles.bursting;
2678            handles.stimulation_data_I = handles.bursting;
2679            set(handles.amplitude1_edit, 'enable', 'on','String', handles.
    amplitude_1)
```

```
2680            set(handles.duration1_edit, 'enable', 'on','String', handles.
    duration_1)
2681            set(handles.biphasic_edit, 'enable', 'on','String', handles.biphasic)
2682            set(handles.delay_edit, 'enable', 'on','String', handles.delay)
2683            set(handles.amplitude2_edit, 'enable', 'off')
2684            set(handles.duration2_edit, 'enable', 'off')
2685            set(handles.decay_edit, 'enable', 'off')
2686            set(handles.resolution_edit, 'enable', 'on','String', handles.
    resolution)
2687            set(handles.repetition_edit, 'enable', 'on','String', handles.
    repetition)
2688            set(handles.Q_exp_edit, 'enable', 'off')
2689            set(handles.Q2_exp_edit, 'enable', 'off')
2690            set(handles.A_exp_edit, 'enable', 'off')
2691            set(handles.A2_exp_edit, 'enable', 'off')
2692            set(handles.B_exp_edit, 'enable', 'off')
2693            set(handles.B2_exp_edit, 'enable', 'off')
2694            set(handles.C_exp_edit, 'enable', 'off')
2695            set(handles.C2_exp_edit, 'enable', 'off')
2696            set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
2697            set(handles.rep_amp_ratio_edit, 'enable', 'on','String', handles.
    rep_amp_ratio)
2698            uicontrol(hObject)
2699            return
2700        end
2701    end
2702
2703
2704    % Attempt Communication with Microcontroller
2705    priorPorts = instrfind; % finds any existing Serial Ports in MATLAB
2706    delete(priorPorts) % and deletes them
2707
2708    % Define Serial Port has the one chosen by user in COM pop up
2709    s = serial(handles.COM_popup_string{handles.COM_popup_value});
2710
2711    % s = serial('COM6'); %assigns the object s to serial port
2712
2713    % Definitions of Serial Communication
2714    set(s, 'InputBufferSize', 1);
2715    set(s, 'FlowControl', 'none');
2716    set(s, 'BaudRate', 38400);
2717    set(s, 'Parity', 'none');
2718    set(s, 'DataBits', 8);
2719    set(s, 'StopBit', 1);
2720    set(s, 'Timeout',100);
2721    %clc;
2722
```

```
% Display of the above definitons in Command Window
% disp(get(s,'Name'));
% prop(1)=(get(s,'BaudRate'));
% prop(2)=(get(s,'DataBits'));
% prop(3)=(get(s, 'StopBit'));
% prop(4)=(get(s, 'InputBufferSize'));
% disp([num2str(prop)]);

% newobjs = instrfind

% tries to open serial port
try
    fopen(s);
catch err % if some error occurs while trying to open serial port
    errordlg('Something went wrong with the connection. Maybe another program using the device? If not, try re-connect the device',...
        'Invalid Input','modal')
    uicontrol(hObject)
    return
end


% Firing Type Sending
send_firing_type = handles.firing_type;
% if user is programming another channel when Alone option is selected and
% has not yet applied the previous programmed stimulation
if send_firing_type == 1 && isempty(find(handles.old_channel == handles.channel_no , 1)) == 1 && handles.number_sent_channels > 1
    final_answer = questdlg('You are changing the channel to use alone. Are you sure?','Firing Channel','Yes','No','No');
    switch final_answer
        case 'Yes'
            fprintf(s, '%d\r', -13) % -13 is the value chosen to control Firing Type in mcu
            pause(0.02);
            fprintf(s, '%d\r', send_firing_type) % sends the Firing Type corresponding number
            pause(0.02);

        case 'No'
            return
    end
% if user is changing the firing type after already uploading a different
% firing type but has not applied it yet
elseif send_firing_type ~= handles.old_firing && (handles.old_firing == 1 || handles.old_firing == 2 || handles.old_firing == 3)
```

```
2763      final_answer = questdlg('You are changing the firing type from the
          definitions already uploaded without applying any stimulus. Are you sure?','
          Firing Channel','Yes','No','No');
2764      switch final_answer
2765          case 'Yes'
2766              fprintf(s, '%d\r', -13) % -13 is the value chosen to control Firing
          Type in mcu
2767              pause(0.02);
2768              fprintf(s, '%d\r', send_firing_type) % sends the Firing Type
          corresponding number
2769              pause(0.02);
2770          case 'No'
2771              return
2772      end
2773  else
2774      fprintf(s, '%d\r', -13) % -13 is the value chosen to control repetition
          amplitude ratio in mcu
2775      pause(0.02);
2776      fprintf(s, '%d\r', send_firing_type) % sends the Firing Type corresponding
          number
2777      pause(0.02);
2778      % to remember the used firing type
2779      handles.old_firing = send_firing_type;
2780  end
2781
2782
2783  % Channel Sending
2784  send_channel = handles.channel_no;
2785  % if user is uploading values for the same channel before applying the one
2786  % previously uploaded
2787  if isempty(find(handles.old_channel == send_channel , 1)) == 0
2788      final_answer = questdlg('You are reprogramming a channel. Are you sure?','
          Channel','Yes','No','No');
2789      switch final_answer
2790          case 'Yes'
2791              fprintf(s, '%d\r', -7) % -7 is the value chosen to control channel in
           mcu
2792              pause(0.02);
2793              fprintf(s, '%d\r', send_channel) % sends the channel number
2794              pause(0.02);
2795          case 'No'
2796              return
2797      end
2798  else
2799      fprintf(s, '%d\r', -7) % -7 is the value chosen to control channel in mcu
2800      pause(0.02);
2801      fprintf(s, '%d\r', send_channel) % sends the channel number
2802      pause(0.02);
```

```
2803      % to remember the previous channel and know how many channels were
2804      % programmed
2805      handles.old_channel(handles.number_sent_channels) = send_channel;
2806      handles.number_sent_channels = handles.number_sent_channels + 1;
2807  end
2808
2809
2810  h = waitbar(0,'Uploading... Closing this window or canceling will reset uploaded
          values and may crash the program!','CreateCancelBtn',...
2811      'setappdata(gcbf,''canceling'',1)');
2812  setappdata(h,'canceling',0);
2813
2814
2815
2816  handles.timescale = (0:(1/handles.time_resolution):(length(handles.current_data_I
          )-1)/handles.time_resolution);
2817  handles.timescale_stimulation = (0:(1/handles.time_resolution):(length(handles.
          stimulation_data_I)-1)/handles.time_resolution);
2818
2819  length(handles.stimulation_data_I);
2820
2821  u = handles.stimulation_data_I;
2822  data=round((u*(4095))/5); % use 3.3 if it is just for testing with KL26
          incorporated DAC
2823
2824  % Waveform Sending
2825  mj = 1;
2826  rep = 0;
2827  for mj=1:length(data)
2828      value = data(mj);
2829
2830      % if user tries to cancel upload
2831      if getappdata(h,'canceling')
2832          fprintf(s, '%d\r', -4)
2833          pause(0.02);
2834          break
2835      end
2836
2837      if(value==0 && rep~=0) %if numbers are zero
2838          fprintf(s, '%d\r', -1)
2839          rep=0;
2840          pause(0.02);
2841      elseif(value>0 && rep~=1) %if numbers return to positive, and only then, then
           is >. As we want only during negative then >=
2842          fprintf(s, '%d\r', -2)
2843          rep=1;
2844          pause(0.02);
```

```
2845      elseif(value<0 && rep~=2) %if numbers return to positive, and only then, then
           is >. As we want only during negative then >=
2846          fprintf(s, '%d\r', -3)
2847          rep=2;
2848          pause(0.02);
2849      end
2850
2851      fprintf(s, '%d\r', abs(value))
2852
2853      pause(0.02);
2854      % a small pause is used because MATLAB was sending it to fast for
2855      % the MCU
2856
2857      waitbar(mj / length(data));
2858  end
2859
2860  % if user pressed 'Cancel' in waitbar
2861  if getappdata(h,'canceling')
2862      errordlg('All settings already uploaded were reseted',...
2863          'Uploading Cancelled','modal')
2864      uicontrol(hObject)
2865      fprintf(s, '%d\r', -14)
2866      pause(0.02);
2867      fprintf(s, '%d\r', -6)
2868      pause(0.02);
2869      delete(h);
2870
2871  else
2872      fprintf(s, '%d\r', -4) % after every sample data sent, -4 acts as a
          terminator
2873
2874      % Repetition Rate Sending
2875      send_repetitions = handles.n_repetitions+1;
2876      send_interstimulus = round(handles.interstimulus_time*handles.time_ms_us);
2877      send_reps_max = handles.reps_max;
2878      fprintf(s, '%d\r', -10) % -10 is the value chosen to control repetition rate
          in mcu
2879      pause(0.02);
2880      fprintf(s, '%d\r', send_repetitions) % sends the  repetition rate
2881      pause(0.02);
2882      fprintf(s, '%d\r', send_interstimulus) % sends the interstimulus time
2883      pause(0.02);
2884      fprintf(s, '%d\r', send_reps_max) % sends the maximum repetitons it can make
2885                                        % before value is gets bigger than
2886                                        % maximum amplitude
2887      pause(0.02);
2888
2889
```

```
2890      % Repetition Amplitude Ratio Sending
2891      send_rep_amp_ratio = handles.rep_amp_ratio;
2892      fprintf(s, '%d\r', -11) % -11 is the value chosen to control repetition
          amplitude ratio in mcu
2893      pause(0.02);
2894      if send_rep_amp_ratio > 0 && send_rep_amp_ratio < 1
2895          fprintf(s, '%d\r', -15) % -15 is the value chosen to signal the mcu that
          repetition amplitude ratio is smaller than 1
2896          pause(0.02);
2897          fprintf(s, '%d\r', ((1/send_rep_amp_ratio)*10)) % sends an inverted
          repetition amplitude ratio multiplied by 10
2898                                                          % so it can send value
          smaller than 1 and with one decimal point
2899          pause(0.02);
2900      else
2901          fprintf(s, '%d\r', -16) % -16 is the value chosen to signal the mcu that
          repetition amplitude ratio is bigger than 1
2902          pause(0.02);
2903          fprintf(s, '%d\r', (send_rep_amp_ratio*10)) % sends the repetition
          amplitude ratio multiplied by 10
2904                                                      % so it can send value
          smaller than 1 and with one decimal point
2905          pause(0.02);
2906      end
2907
2908
2909
2910      % Time Lag Sending
2911      send_time_lag = handles.time_lag;
2912      fprintf(s, '%d\r', -12) % -11 is the value chosen to control repetition
          amplitude ratio in mcu
2913      pause(0.02);
2914      fprintf(s, '%d\r', send_time_lag) % sends the repetition amplitude ratio
2915      pause(0.02);
2916
2917
2918      % End of Transmission - however transmission is only really finished
2919      % with 'Apply' or 'Reset'
2920
2921  end
2922  delete(h); % close waitbat
2923  fclose(s); %close the serial port
2924  delete(s) % deleting and cleaning of variable. it is more efective
2925  clear s;
2926
2927  guidata(hObject, handles);
2928
2929
```

```
2930
2931 function Q_exp_edit_Callback(hObject, eventdata, handles)
2932 % hObject    handle to Q_exp_edit (see GCBO)
2933 % eventdata  reserved - to be defined in a future version of MATLAB
2934 % handles    structure with handles and user data (see GUIDATA)
2935
2936 % Hints: get(hObject,'String') returns contents of Q_exp_edit as text
2937 %        str2double(get(hObject,'String')) returns contents of Q_exp_edit as a
     double
2938
2939 q_exp_input = str2double(get(hObject,'string'));
2940
2941 if isfield(handles, 'plot_popup_value' )
2942     % handles.firing_popup_value exists - you may access it
2943 else
2944     % handles.firing_popup_value does not exist
2945     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
2946     uicontrol(hObject)
2947     return
2948 end
2949
2950 if isnan(q_exp_input) || q_exp_input > 1
2951     errordlg('You must enter a numeric value smaller or equal to 1','Invalid
     Input','modal')
2952     uicontrol(hObject)
2953     return
2954 else
2955     handles.q = q_exp_input;
2956     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
     duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
2957     handles.timescale = (0:(1/handles.time_resolution):(handles.
     stimulation_fulltime+handles.t_f));
2958     value = handles.plot_popup_value;
2959     string = handles.plot_popup_string;
2960     switch string{value}
2961         case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
2962             handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
     handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
2963             i=1;
2964             total_decay_amplitude = 0;
2965             for i=1:handles.decay_steps
2966                 handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
     amplitude_1*handles.q*...
2967                     exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
     .*...
2968                     ones(1,handles.resolution*handles.time_resolution));
2969                 total_decay_amplitude = total_decay_amplitude + (handles.
     amplitude_1*handles.q*...
```

```
2970                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
2971            end
2972            amplitude_CB = (handles.amplitude_1*handles.duration_1 +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
2973            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
2974            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
2975            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
2976            handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
    delay*handles.time_resolution))...
2977                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
2978                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
2979                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
2980            handles.current_data_I = handles.quasi_trap;
2981            handles.stimulation_data_I = handles.quasi_trap;
2982    end
2983
2984    handles.reps_max = 0;
2985    if handles.repetition ~= 0
2986        for n_reps = 1:handles.n_repetitions
2987            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
2988                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
2989                handles.reps_max = n_reps;
2990            end
2991            if handles.reps_max ~=0
2992                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
2993            else
2994                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
2995            end
2996        end
2997    end
2998
2999    if handles.fullstimulation_duration < handles.stimulation_fulltime
```

```
3000         set(handles.stim_duration_edit, 'enable', 'on','String', handles.
         stimulation_fulltime)
3001     end
3002
3003 end
3004
3005 guidata(hObject, handles);
3006
3007
3008 % --- Executes during object creation, after setting all properties.
3009 function Q_exp_edit_CreateFcn(hObject, eventdata, handles)
3010 % hObject    handle to Q_exp_edit (see GCBO)
3011 % eventdata  reserved - to be defined in a future version of MATLAB
3012 % handles    empty - handles not created until after all CreateFcns called
3013
3014 % Hint: edit controls usually have a white background on Windows.
3015 %       See ISPC and COMPUTER.
3016 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
         defaultUicontrolBackgroundColor'))
3017     set(hObject,'BackgroundColor','white');
3018 end
3019
3020
3021
3022 function Q2_exp_edit_Callback(hObject, eventdata, handles)
3023 % hObject    handle to Q2_exp_edit (see GCBO)
3024 % eventdata  reserved - to be defined in a future version of MATLAB
3025 % handles    structure with handles and user data (see GUIDATA)
3026
3027 % Hints: get(hObject,'String') returns contents of Q2_exp_edit as text
3028 %        str2double(get(hObject,'String')) returns contents of Q2_exp_edit as a
         double
3029 q2_exp_input = str2double(get(hObject,'string'));
3030
3031 if isfield(handles, 'plot_popup_value' )
3032     % handles.firing_popup_value exists - you may access it
3033 else
3034     % handles.firing_popup_value does not exist
3035     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3036     uicontrol(hObject)
3037     return
3038 end
3039
3040 if isnan(q2_exp_input)
3041     errordlg('You must enter a numeric value','Invalid Input','modal')
3042     uicontrol(hObject)
3043     return
3044 else
```

```
3045    handles.q2 = q2_exp_input;
3046    handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
        duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
3047    handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
3048    value = handles.plot_popup_value;
3049    string = handles.plot_popup_string;
3050    switch string{value}
3051        case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
3052            handles.quasi_trap = [zeros(1,handles.t_i*handles.time_resolution) (-
        handles.amplitude_1).*ones(1,handles.duration_1*handles.time_resolution)];
3053            i=1;
3054            total_decay_amplitude = 0;
3055            for i=1:handles.decay_steps
3056                handles.quasi_trap = horzcat(handles.quasi_trap,(-handles.
        amplitude_1*handles.q*...
3057                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s))
        .*...
3058                    ones(1,handles.resolution*handles.time_resolution));
3059                total_decay_amplitude = total_decay_amplitude + (handles.
        amplitude_1*handles.q*...
3060                    exp(-handles.q2*(i*handles.resolution)*handles.time_ms_s));
3061            end
3062            amplitude_CB = (handles.amplitude_1*handles.duration_1 +
        total_decay_amplitude*handles.resolution)/handles.biphasic;
3063            handles.duration_balance_total = handles.biphasic.*(handles.
        duration_1 + handles.decay);
3064            handles.stimulation_fulltime = handles.duration_1+handles.
        duration_balance_total+handles.delay+handles.decay;
3065            handles.timescale = (0:(1/handles.time_resolution):(handles.
        stimulation_fulltime+handles.t_f));
3066            handles.quasi_trap = horzcat(handles.quasi_trap, [zeros(1,(handles.
        delay*handles.time_resolution))...
3067                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
        time_resolution)...
3068                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
        time_resolution)...
3069                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
3070            handles.current_data_I = handles.quasi_trap;
3071            handles.stimulation_data_I = handles.quasi_trap;
3072    end
3073
3074    handles.reps_max = 0;
3075    if handles.repetition ~= 0
3076        for n_reps = 1:handles.n_repetitions
```

```
3077            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
        amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
         handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
        reps_max == 0)...
3078                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
         <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
        DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
        +1)) ) && handles.reps_max == 0)
3079                handles.reps_max = n_reps;
3080            end
3081            if handles.reps_max ~=0
3082                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
        handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
        .^(handles.reps_max)).*handles.stimulation_data_I);
3083            else
3084                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
        handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
        .^(n_reps)).*handles.stimulation_data_I);
3085            end
3086        end
3087    end
3088
3089    if handles.fullstimulation_duration < handles.stimulation_fulltime
3090        set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
3091    end
3092
3093 end
3094
3095
3096 % --- Executes during object creation, after setting all properties.
3097 function Q2_exp_edit_CreateFcn(hObject, eventdata, handles)
3098 % hObject    handle to Q2_exp_edit (see GCBO)
3099 % eventdata  reserved - to be defined in a future version of MATLAB
3100 % handles    empty - handles not created until after all CreateFcns called
3101
3102 % Hint: edit controls usually have a white background on Windows.
3103 %       See ISPC and COMPUTER.
3104 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
3105     set(hObject,'BackgroundColor','white');
3106 end
3107
3108
3109
3110 function A_exp_edit_Callback(hObject, eventdata, handles)
3111 % hObject    handle to A_exp_edit (see GCBO)
3112 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of A_exp_edit as text
%        str2double(get(hObject,'String')) returns contents of A_exp_edit as a double

a_exp_input = str2double(get(hObject,'string'));

if isfield(handles, 'plot_popup_value' )
    % handles.firing_popup_value exists - you may access it
else
    % handles.firing_popup_value does not exist
    errordlg('You must choose a Waverform Type!','Invalid Input','modal')
    uicontrol(hObject)
    return
end

if isnan(a_exp_input)
    errordlg('You must enter a numeric value','Invalid Input','modal')
    uicontrol(hObject)
    return
elseif (a_exp_input + handles.b + handles.c) > 1 % This condition was created because if
    % the sum of these 3 parameters is bigger than 1, the wave as a step
    % between end of increase and begin of decay
    errordlg('Sum of A, B and C cannot be bigger than 1','Invalid Input','modal')
    uicontrol(hObject)
    return
else
    handles.a = a_exp_input;
    handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
    handles.timescale = (0:(1/handles.time_resolution):(handles.stimulation_fulltime+handles.t_f));
    value = handles.plot_popup_value;
    string = handles.plot_popup_string;
    switch string{value}
        case 'Tri-Exponential' % User selects tri-exponential
            handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
            j_i = 1;
            j_d = 1;
            total_increase_amplitude = 0;
            total_decay_amplitude = 0;
            for j_i=1:handles.increase_steps
                handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.resolution*handles.time_resolution)...
```

```
3154                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
3155                    ones(1,handles.resolution*handles.time_resolution));
3156                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
3157                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
3158            end
3159            for j_d=1:handles.decay_steps
3160                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3161                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3162                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3163                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
3164                    ones(1,handles.resolution*handles.time_resolution));
3165                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3166                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3167                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3168                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
3169            end
3170            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
3171            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
3172            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
3173            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
3174            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
3175                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
3176                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
3177                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
3178            handles.current_data_I = handles.tri_exp;
3179            handles.stimulation_data_I = handles.tri_exp;
3180    end
3181
3182    handles.reps_max = 0;
```

```
3183      if handles.repetition ~= 0
3184          for n_reps = 1:handles.n_repetitions
3185              if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
          amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
           handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
          reps_max == 0)...
3186                      || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
           <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
          DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
          +1)) ) && handles.reps_max == 0)
3187                  handles.reps_max = n_reps;
3188              end
3189              if handles.reps_max ~=0
3190                  handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
          handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
          .^(handles.reps_max)).*handles.stimulation_data_I);
3191              else
3192                  handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
          handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
          .^(n_reps)).*handles.stimulation_data_I);
3193              end
3194          end
3195      end
3196
3197      if handles.fullstimulation_duration < handles.stimulation_fulltime
3198          set(handles.stim_duration_edit, 'enable', 'on','String', handles.
          stimulation_fulltime)
3199      end
3200
3201  end
3202
3203  guidata(hObject, handles);
3204
3205
3206  % --- Executes during object creation, after setting all properties.
3207  function A_exp_edit_CreateFcn(hObject, eventdata, handles)
3208  % hObject    handle to A_exp_edit (see GCBO)
3209  % eventdata  reserved - to be defined in a future version of MATLAB
3210  % handles    empty - handles not created until after all CreateFcns called
3211
3212  % Hint: edit controls usually have a white background on Windows.
3213  %       See ISPC and COMPUTER.
3214  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
          defaultUicontrolBackgroundColor'))
3215      set(hObject,'BackgroundColor','white');
3216  end
3217
3218
```

```
3219
3220 function A2_exp_edit_Callback(hObject, eventdata, handles)
3221 % hObject    handle to A2_exp_edit (see GCBO)
3222 % eventdata  reserved - to be defined in a future version of MATLAB
3223 % handles    structure with handles and user data (see GUIDATA)
3224
3225 % Hints: get(hObject,'String') returns contents of A2_exp_edit as text
3226 %        str2double(get(hObject,'String')) returns contents of A2_exp_edit as a
     double
3227
3228 a2_exp_input = str2double(get(hObject,'string'));
3229
3230 if isfield(handles, 'plot_popup_value' )
3231     % handles.firing_popup_value exists - you may access it
3232 else
3233     % handles.firing_popup_value does not exist
3234     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3235     uicontrol(hObject)
3236     return
3237 end
3238
3239 if isnan(a2_exp_input)
3240     errordlg('You must enter a numeric value','Invalid Input','modal')
3241     uicontrol(hObject)
3242     return
3243 else
3244     handles.a2 = a2_exp_input;
3245     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
     duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
3246     handles.timescale = (0:(1/handles.time_resolution):(handles.
     stimulation_fulltime+handles.t_f));
3247     value = handles.plot_popup_value;
3248     string = handles.plot_popup_string;
3249     switch string{value}
3250         case 'Tri-Exponential' % User selects tri-exponential
3251             handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
3252             j_i = 1;
3253             j_d = 1;
3254             total_increase_amplitude = 0;
3255             total_decay_amplitude = 0;
3256             for j_i=1:handles.increase_steps
3257                 handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
     amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
     resolution*handles.time_resolution)...
3258                     +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
     handles.t_i)-handles.t_i)))/handles.time_resolution).*...
3259                     ones(1,handles.resolution*handles.time_resolution));
```

```
3260                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
3261                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
3262            end
3263            for j_d=1:handles.decay_steps
3264                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3265                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3266                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3267                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
3268                    ones(1,handles.resolution*handles.time_resolution));
3269                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3270                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3271                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3272                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
3273            end
3274            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
3275            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
3276            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
3277            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
3278            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
3279                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
3280                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
3281                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
3282            handles.current_data_I = handles.tri_exp;
3283            handles.stimulation_data_I = handles.tri_exp;
3284        end
3285
3286        handles.reps_max = 0;
3287        if handles.repetition ~= 0
3288            for n_reps = 1:handles.n_repetitions
```

```
3289            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
3290                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
3291                    handles.reps_max = n_reps;
3292            end
3293            if handles.reps_max ~=0
3294                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
3295            else
3296                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
3297            end
3298        end
3299    end
3300
3301    if handles.fullstimulation_duration < handles.stimulation_fulltime
3302        set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
3303    end
3304
3305 end
3306
3307 guidata(hObject, handles);
3308
3309
3310 % --- Executes during object creation, after setting all properties.
3311 function A2_exp_edit_CreateFcn(hObject, eventdata, handles)
3312 % hObject    handle to A2_exp_edit (see GCBO)
3313 % eventdata  reserved - to be defined in a future version of MATLAB
3314 % handles    empty - handles not created until after all CreateFcns called
3315
3316 % Hint: edit controls usually have a white background on Windows.
3317 %       See ISPC and COMPUTER.
3318 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
3319     set(hObject,'BackgroundColor','white');
3320 end
3321
3322
3323
3324 function B_exp_edit_Callback(hObject, eventdata, handles)
```

```
3325 % hObject    handle to B_exp_edit (see GCBO)
3326 % eventdata  reserved - to be defined in a future version of MATLAB
3327 % handles    structure with handles and user data (see GUIDATA)
3328
3329 % Hints: get(hObject,'String') returns contents of B_exp_edit as text
3330 %        str2double(get(hObject,'String')) returns contents of B_exp_edit as a
     double
3331
3332 b_exp_input = str2double(get(hObject,'string'));
3333
3334 if isfield(handles, 'plot_popup_value' )
3335     % handles.firing_popup_value exists - you may access it
3336 else
3337     % handles.firing_popup_value does not exist
3338     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3339     uicontrol(hObject)
3340     return
3341 end
3342
3343 if isnan(b_exp_input)
3344     errordlg('You must enter a numeric value','Invalid Input','modal')
3345     uicontrol(hObject)
3346     return
3347 elseif (b_exp_input + handles.a + handles.c) > 1
3348     errordlg('Sum of A, B and C cannot be bigger than 1','Invalid Input','modal')
3349     uicontrol(hObject)
3350     return
3351 else
3352     handles.b = b_exp_input;
3353     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
     duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
3354     handles.timescale = (0:(1/handles.time_resolution):(handles.
     stimulation_fulltime+handles.t_f));
3355     value = handles.plot_popup_value;
3356     string = handles.plot_popup_string;
3357     switch string{value}
3358         case 'Tri-Exponential' % User selects tri-exponential
3359             handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
3360             j_i = 1;
3361             j_d = 1;
3362             total_increase_amplitude = 0;
3363             total_decay_amplitude = 0;
3364             for j_i=1:handles.increase_steps
3365                 handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
     amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
     resolution*handles.time_resolution)...
3366                     +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
     handles.t_i)-handles.t_i)))/handles.time_resolution).*...
```

```
3367                    ones(1,handles.resolution*handles.time_resolution));
3368                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
3369                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
3370            end
3371            for j_d=1:handles.decay_steps
3372                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3373                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3374                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3375                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
3376                    ones(1,handles.resolution*handles.time_resolution));
3377                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3378                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3379                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3380                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
3381            end
3382            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
3383            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
3384            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
3385            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
3386            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
3387                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
3388                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
3389                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
3390            handles.current_data_I = handles.tri_exp;
3391            handles.stimulation_data_I = handles.tri_exp;
3392        end
3393
3394        handles.reps_max = 0;
3395        if handles.repetition ~= 0
3396            for n_reps = 1:handles.n_repetitions
```

```
3397            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
3398                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)                    handles.reps_max = n_reps
    ;
3399            end
3400            if handles.reps_max ~=0
3401                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
3402            else
3403                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
3404            end
3405        end
3406    end
3407
3408    if handles.fullstimulation_duration < handles.stimulation_fulltime
3409        set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
3410    end
3411
3412 end
3413
3414 guidata(hObject, handles);
3415
3416
3417
3418 % --- Executes during object creation, after setting all properties.
3419 function B_exp_edit_CreateFcn(hObject, eventdata, handles)
3420 % hObject    handle to B_exp_edit (see GCBO)
3421 % eventdata  reserved - to be defined in a future version of MATLAB
3422 % handles    empty - handles not created until after all CreateFcns called
3423
3424 % Hint: edit controls usually have a white background on Windows.
3425 %       See ISPC and COMPUTER.
3426 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
3427     set(hObject,'BackgroundColor','white');
3428 end
3429
3430
3431
```

```
3432 function B2_exp_edit_Callback(hObject, eventdata, handles)
3433 % hObject    handle to B2_exp_edit (see GCBO)
3434 % eventdata  reserved - to be defined in a future version of MATLAB
3435 % handles    structure with handles and user data (see GUIDATA)
3436
3437 % Hints: get(hObject,'String') returns contents of B2_exp_edit as text
3438 %        str2double(get(hObject,'String')) returns contents of B2_exp_edit as a
     double
3439
3440 b2_exp_input = str2double(get(hObject,'string'));
3441
3442 if isfield(handles, 'plot_popup_value' )
3443     % handles.firing_popup_value exists - you may access it
3444 else
3445     % handles.firing_popup_value does not exist
3446     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3447     uicontrol(hObject)
3448     return
3449 end
3450
3451 if isnan(b2_exp_input)
3452     errordlg('You must enter a numeric value','Invalid Input','modal')
3453     uicontrol(hObject)
3454     return
3455 else
3456     handles.b2 = b2_exp_input;
3457     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
     duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
3458     handles.timescale = (0:(1/handles.time_resolution):(handles.
     stimulation_fulltime+handles.t_f));
3459     value = handles.plot_popup_value;
3460     string = handles.plot_popup_string;
3461     switch string{value}
3462         case 'Tri-Exponential' % User selects tri-exponential
3463             handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
3464             j_i = 1;
3465             j_d = 1;
3466             total_increase_amplitude = 0;
3467             total_decay_amplitude = 0;
3468             for j_i=1:handles.increase_steps
3469                 handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
     amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
     resolution*handles.time_resolution)...
3470                     +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
     handles.t_i)-handles.t_i)))/handles.time_resolution).*...
3471                     ones(1,handles.resolution*handles.time_resolution));
```

```
3472                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
3473                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
3474            end
3475            for j_d=1:handles.decay_steps
3476                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3477                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3478                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3479                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
3480                    ones(1,handles.resolution*handles.time_resolution));
3481                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3482                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3483                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3484                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
3485            end
3486            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
3487            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
3488            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
3489            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
3490            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
3491                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
3492                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
3493                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
3494            handles.current_data_I = handles.tri_exp;
3495            handles.stimulation_data_I = handles.tri_exp;
3496        end
3497
3498        handles.reps_max = 0;
3499        if handles.repetition ~= 0
3500            for n_reps = 1:handles.n_repetitions
```

```
3501            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
    amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
     handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
    reps_max == 0)...
3502                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
     <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
    +1)) ) && handles.reps_max == 0)
3503                handles.reps_max = n_reps;
3504            end
3505            if handles.reps_max ~=0
3506                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(handles.reps_max)).*handles.stimulation_data_I);
3507            else
3508                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
    handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
    .^(n_reps)).*handles.stimulation_data_I);
3509            end
3510        end
3511    end
3512
3513    if handles.fullstimulation_duration < handles.stimulation_fulltime
3514        set(handles.stim_duration_edit, 'enable', 'on','String', handles.
    stimulation_fulltime)
3515    end
3516
3517 end
3518
3519 guidata(hObject, handles);
3520
3521
3522
3523 % --- Executes during object creation, after setting all properties.
3524 function B2_exp_edit_CreateFcn(hObject, eventdata, handles)
3525 % hObject    handle to B2_exp_edit (see GCBO)
3526 % eventdata  reserved - to be defined in a future version of MATLAB
3527 % handles    empty - handles not created until after all CreateFcns called
3528
3529 % Hint: edit controls usually have a white background on Windows.
3530 %       See ISPC and COMPUTER.
3531 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
3532     set(hObject,'BackgroundColor','white');
3533 end
3534
3535
3536
```

```
3537 function C_exp_edit_Callback(hObject, eventdata, handles)
3538 % hObject    handle to C_exp_edit (see GCBO)
3539 % eventdata  reserved - to be defined in a future version of MATLAB
3540 % handles    structure with handles and user data (see GUIDATA)
3541
3542 % Hints: get(hObject,'String') returns contents of C_exp_edit as text
3543 %        str2double(get(hObject,'String')) returns contents of C_exp_edit as a
         double
3544
3545 c_exp_input = str2double(get(hObject,'string'));
3546
3547 if isfield(handles, 'plot_popup_value' )
3548     % handles.firing_popup_value exists - you may access it
3549 else
3550     % handles.firing_popup_value does not exist
3551     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3552     uicontrol(hObject)
3553     return
3554 end
3555
3556 if isnan(c_exp_input)
3557     errordlg('You must enter a numeric value','Invalid Input','modal')
3558     uicontrol(hObject)
3559     return
3560 elseif (c_exp_input + handles.a + handles.b) > 1
3561     errordlg('Sum of A, B and C cannot be bigger than 1','Invalid Input','modal')
3562     uicontrol(hObject)
3563     return
3564 else
3565     handles.c = c_exp_input;
3566     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
         duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
3567     handles.timescale = (0:(1/handles.time_resolution):(handles.
         stimulation_fulltime+handles.t_f));
3568     value = handles.plot_popup_value;
3569     string = handles.plot_popup_string;
3570     switch string{value}
3571         case 'Tri-Exponential' % User selects tri-exponential
3572             handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
3573             j_i = 1;
3574             j_d = 1;
3575             total_increase_amplitude = 0;
3576             total_decay_amplitude = 0;
3577             for j_i=1:handles.increase_steps
3578                 handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
         amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
         resolution*handles.time_resolution)...
```

```
3579                    +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i)))/handles.time_resolution).*...
3580                    ones(1,handles.resolution*handles.time_resolution));
3581                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
3582                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
3583            end
3584            for j_d=1:handles.decay_steps
3585                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3586                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3587                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3588                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
3589                    ones(1,handles.resolution*handles.time_resolution));
3590                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3591                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3592                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3593                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
3594            end
3595            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
3596            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
3597            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
3598            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
3599            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
3600                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
3601                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
3602                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
3603            handles.current_data_I = handles.tri_exp;
3604            handles.stimulation_data_I = handles.tri_exp;
3605        end
3606
3607        handles.reps_max = 0;
```

```
3608      if handles.repetition ~= 0
3609          for n_reps = 1:handles.n_repetitions
3610              if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
          amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
           handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
          reps_max == 0)...
3611                      || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
           <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
          DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
          +1)) ) && handles.reps_max == 0)
3612                  handles.reps_max = n_reps;
3613              end
3614              if handles.reps_max ~=0
3615                  handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
          handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
          .^(handles.reps_max)).*handles.stimulation_data_I);
3616              else
3617                  handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
          handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
          .^(n_reps)).*handles.stimulation_data_I);
3618              end
3619          end
3620      end
3621
3622      if handles.fullstimulation_duration < handles.stimulation_fulltime
3623          set(handles.stim_duration_edit, 'enable', 'on','String', handles.
          stimulation_fulltime)
3624      end
3625
3626  end
3627
3628  guidata(hObject, handles);
3629
3630
3631
3632  % --- Executes during object creation, after setting all properties.
3633  function C_exp_edit_CreateFcn(hObject, eventdata, handles)
3634  % hObject    handle to C_exp_edit (see GCBO)
3635  % eventdata  reserved - to be defined in a future version of MATLAB
3636  % handles    empty - handles not created until after all CreateFcns called
3637
3638  % Hint: edit controls usually have a white background on Windows.
3639  %       See ISPC and COMPUTER.
3640  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
          defaultUicontrolBackgroundColor'))
3641      set(hObject,'BackgroundColor','white');
3642  end
3643
```

```
3644
3645
3646 function C2_exp_edit_Callback(hObject, eventdata, handles)
3647 % hObject    handle to C2_exp_edit (see GCBO)
3648 % eventdata  reserved - to be defined in a future version of MATLAB
3649 % handles    structure with handles and user data (see GUIDATA)
3650
3651 % Hints: get(hObject,'String') returns contents of C2_exp_edit as text
3652 %        str2double(get(hObject,'String')) returns contents of C2_exp_edit as a
     double
3653
3654 c2_exp_input = str2double(get(hObject,'string'));
3655
3656 if isfield(handles, 'plot_popup_value' )
3657     % handles.firing_popup_value exists - you may access it
3658 else
3659     % handles.firing_popup_value does not exist
3660     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3661     uicontrol(hObject)
3662     return
3663 end
3664
3665 if isnan(c2_exp_input)
3666     errordlg('You must enter a numeric value','Invalid Input','modal')
3667     uicontrol(hObject)
3668     return
3669 else
3670     handles.c2 = c2_exp_input;
3671     handles.stimulation_fulltime = handles.duration_1+handles.duration_2+handles.
     duration_balance_1+handles.duration_balance_2+handles.delay+handles.decay;
3672     handles.timescale = (0:(1/handles.time_resolution):(handles.
     stimulation_fulltime+handles.t_f));
3673     value = handles.plot_popup_value;
3674     string = handles.plot_popup_string;
3675     switch string{value}
3676         case 'Tri-Exponential' % User selects tri-exponential
3677             handles.tri_exp = zeros(1,handles.t_i*handles.time_resolution);
3678             j_i = 1;
3679             j_d = 1;
3680             total_increase_amplitude = 0;
3681             total_decay_amplitude = 0;
3682             for j_i=1:handles.increase_steps
3683                 handles.tri_exp = horzcat(handles.tri_exp, ((((-handles.
     amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i))*(j_i*handles.
     resolution*handles.time_resolution)...
3684                     +(-((-handles.amplitude_1*handles.t_i)/(handles.duration_1+
     handles.t_i)-handles.t_i)))/handles.time_resolution).*...
3685                     ones(1,handles.resolution*handles.time_resolution));
```

```
3686                total_increase_amplitude = total_increase_amplitude + (((handles.
    amplitude_1)/((handles.duration_1+handles.t_i)-handles.t_i)*(j_i*handles.
    resolution*handles.time_resolution)...
3687                    -((handles.amplitude_1*handles.t_i)/(handles.duration_1+
    handles.t_i)-handles.t_i))/handles.time_resolution);
3688            end
3689            for j_d=1:handles.decay_steps
3690                handles.tri_exp = horzcat(handles.tri_exp,(handles.tri_exp((
    handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3691                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3692                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3693                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    ).*...
3694                    ones(1,handles.resolution*handles.time_resolution));
3695                total_decay_amplitude = total_decay_amplitude + (handles.tri_exp
    ((handles.t_i+handles.duration_1)*handles.time_resolution)*((handles.a*...
3696                    exp(-handles.a2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.b*...
3697                    exp(-handles.b2*(j_d*handles.resolution)*handles.time_ms_s))
    +(handles.c*...
3698                    exp(-handles.c2*(j_d*handles.resolution)*handles.time_ms_s)))
    );
3699            end
3700            amplitude_CB = (total_increase_amplitude*handles.resolution +
    total_decay_amplitude*handles.resolution)/handles.biphasic;
3701            handles.duration_balance_total = handles.biphasic.*(handles.
    duration_1 + handles.decay);
3702            handles.stimulation_fulltime = handles.duration_1+handles.
    duration_balance_total+handles.delay+handles.decay;
3703            handles.timescale = (0:(1/handles.time_resolution):(handles.
    stimulation_fulltime+handles.t_f));
3704            handles.tri_exp = horzcat(handles.tri_exp, [zeros(1,(handles.delay*
    handles.time_resolution))...
3705                amplitude_CB.*ones(1,handles.duration_balance_total*handles.
    time_resolution)...
3706                zeros(1,floor((length(handles.timescale)-handles.t_i*handles.
    time_resolution)...
3707                -((handles.stimulation_fulltime)*handles.time_resolution)))]);
3708            handles.current_data_I = handles.tri_exp;
3709            handles.stimulation_data_I = handles.tri_exp;
3710    end
3711
3712    handles.reps_max = 0;
3713    if handles.repetition ~= 0
3714        for n_reps = 1:handles.n_repetitions
```

```
3715            if (handles.amplitude_max ~=0 && (handles.amplitude_max <= handles.
        amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.amplitude_max <=
         handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) ) && handles.
        reps_max == 0)...
3716                    || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
         <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
        DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
        +1)) ) && handles.reps_max == 0)
3717                handles.reps_max = n_reps;
3718            end
3719            if handles.reps_max ~=0
3720                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
        handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
        .^(handles.reps_max)).*handles.stimulation_data_I);
3721            else
3722                handles.current_data_I = horzcat(handles.current_data_I,zeros(1,(
        handles.interstimulus_time*handles.time_resolution)),((handles.rep_amp_ratio)
        .^(n_reps)).*handles.stimulation_data_I);
3723            end
3724        end
3725    end
3726
3727    if handles.fullstimulation_duration < handles.stimulation_fulltime
3728        set(handles.stim_duration_edit, 'enable', 'on','String', handles.
        stimulation_fulltime)
3729    end
3730
3731 end
3732
3733 guidata(hObject, handles);
3734
3735
3736
3737 % --- Executes during object creation, after setting all properties.
3738 function C2_exp_edit_CreateFcn(hObject, eventdata, handles)
3739 % hObject    handle to C2_exp_edit (see GCBO)
3740 % eventdata  reserved - to be defined in a future version of MATLAB
3741 % handles    empty - handles not created until after all CreateFcns called
3742
3743 % Hint: edit controls usually have a white background on Windows.
3744 %       See ISPC and COMPUTER.
3745 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
3746     set(hObject,'BackgroundColor','white');
3747 end
3748
3749
3750
```

```
3751 function stim_duration_edit_Callback(hObject, eventdata, handles)
3752 % hObject    handle to stim_duration_edit (see GCBO)
3753 % eventdata  reserved - to be defined in a future version of MATLAB
3754 % handles    structure with handles and user data (see GUIDATA)
3755
3756 % Hints: get(hObject,'String') returns contents of stim_duration_edit as text
3757 %        str2double(get(hObject,'String')) returns contents of stim_duration_edit
      as a double
3758
3759 stim_duration_input = str2double(get(hObject,'string'));
3760
3761 if isfield(handles, 'plot_popup_value' )
3762     % handles.firing_popup_value exists - you may access it
3763 else
3764     % handles.firing_popup_value does not exist
3765     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3766     uicontrol(hObject)
3767     return
3768 end
3769
3770 if isnan(stim_duration_input) || stim_duration_input == 0
3771     errordlg('You must enter a numeric value bigger than zero','Invalid Input','
        modal')
3772     uicontrol(hObject)
3773     return
3774 elseif stim_duration_input < handles.stimulation_fulltime
3775     errordlg('You must enter a value equal or bigger than the duration of a
        single pulse','Invalid Input','modal')
3776     uicontrol(hObject)
3777     return
3778 elseif handles.repetition ~= 0 && (1/(handles.repetition)) > stim_duration_input
3779     warndlg('Repetition Cycle bigger than Stimulation Duration! No Repetition
        will be made.','Value Selection', 'modal');
3780     uicontrol(hObject)
3781     return
3782 else
3783     handles.fullstimulation_duration = stim_duration_input;
3784     value = handles.plot_popup_value;
3785     string = handles.plot_popup_string;
3786     handles.reps_max = 0;
3787     n_reps = 1;
3788     handles.interstimulus_time = (1/(handles.repetition));
3789     handles.n_repetitions = (floor(handles.fullstimulation_duration/(handles.
        stimulation_fulltime+handles.interstimulus_time))-1);
3790     switch string{value}
3791         case 'Bi-level Lilly' % User selects bilevel Lilly
3792             handles.current_data_I = handles.bi_level;
3793             handles.stimulation_data_I = handles.bi_level;
```

```
3794            for n_reps = 1:handles.n_repetitions
3795                if handles.amplitude_max ~=0 && handles.reps_max == 0 && (handles
    .amplitude_max <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps)) ||
    handles.amplitude_max <= handles.amplitude_2*((handles.rep_amp_ratio).^(
    n_reps)))
3796                    handles.reps_max = n_reps;
3797                end
3798                if handles.reps_max ~=0
3799                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.bi_level);
3800                else
3801                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.bi_level);
3802                end
3803            end
3804        case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
3805            handles.current_data_I = handles.quasi_trap;
3806            handles.stimulation_data_I = handles.quasi_trap;
3807            for n_reps = 1:handles.n_repetitions
3808                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
3809                        || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
3810                    handles.reps_max = n_reps;
3811                end
3812                if handles.reps_max ~=0
3813                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.quasi_trap);
3814                else
3815                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.quasi_trap);
3816                end
3817            end
3818        case 'Tri-Exponential' % User selects tri-exponential
3819            handles.current_data_I = handles.tri_exp;
3820            handles.stimulation_data_I = handles.tri_exp;
3821            for n_reps = 1:handles.n_repetitions
3822                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
```

```
3823                        || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
3824                    handles.reps_max = n_reps;
3825                end
3826                if handles.reps_max ~=0
3827                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.tri_exp);
3828                else
3829                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.tri_exp);
3830                end
3831            end
3832        case 'Bursting' % User selects bursting strategies
3833            handles.current_data_I = handles.bursting;
3834            handles.stimulation_data_I = handles.bursting;
3835            for n_reps = 1:handles.n_repetitions
3836                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
3837                        || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
3838                    handles.reps_max = n_reps;
3839                end
3840                if handles.reps_max ~=0
3841                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.bursting);
3842                else
3843                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.bursting);
3844                end
3845            end
3846        case 'Rectangular' % User selects pulse
3847            handles.current_data_I = handles.pulse;
3848            handles.stimulation_data_I = handles.pulse;
3849            for n_reps = 1:handles.n_repetitions
3850                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
3851                        || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
3852                    handles.reps_max = n_reps;
```

```
3853                end
3854                if handles.reps_max ~=0
3855                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.pulse);
3856                else
3857                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.pulse);
3858                end
3859            end
3860        case 'Load Waveform'
3861            handles.current_data_I = handles.current_data_I;
3862            handles.stimulation_data_I = handles.current_data_I;
3863            for n_reps = 1:handles.n_repetitions
3864                if (handles.amplitude_max ~=0 && (handles.amplitude_max <= max(
    handles.stimulation_data_I)*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    amplitude_max <= abs(min(handles.stimulation_data_I))*((handles.rep_amp_ratio
    ).^(n_reps+1))) && handles.reps_max == 0)...
3865                        || (handles.amplitude_max ==0 && (handles.
    DAC_amplitude_limit <= max(handles.stimulation_data_I)*((handles.
    rep_amp_ratio).^(n_reps+1)) || handles.DAC_amplitude_limit <= abs(min(handles
    .stimulation_data_I))*((handles.rep_amp_ratio).^(n_reps+1))) && handles.
    reps_max == 0)
3866                    handles.reps_max = n_reps;
3867                end
3868                if handles.reps_max ~=0
3869                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.stimulation_data_I);
3870                else
3871                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.stimulation_data_I);
3872                end
3873            end
3874    end
3875
3876 end
3877
3878 guidata(hObject, handles);
3879
3880
3881
3882
3883 % --- Executes during object creation, after setting all properties.
3884 function stim_duration_edit_CreateFcn(hObject, eventdata, handles)
3885 % hObject    handle to stim_duration_edit (see GCBO)
```

```
3886 % eventdata  reserved - to be defined in a future version of MATLAB
3887 % handles    empty - handles not created until after all CreateFcns called
3888
3889 % Hint: edit controls usually have a white background on Windows.
3890 %       See ISPC and COMPUTER.
3891 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
         defaultUicontrolBackgroundColor'))
3892     set(hObject,'BackgroundColor','white');
3893 end
3894
3895
3896
3897 function rep_amp_ratio_edit_Callback(hObject, eventdata, handles)
3898 % hObject    handle to rep_amp_ratio_edit (see GCBO)
3899 % eventdata  reserved - to be defined in a future version of MATLAB
3900 % handles    structure with handles and user data (see GUIDATA)
3901
3902 % Hints: get(hObject,'String') returns contents of rep_amp_ratio_edit as text
3903 %        str2double(get(hObject,'String')) returns contents of rep_amp_ratio_edit
          as a double
3904
3905 rep_amp_ratio_input = str2double(get(hObject,'string'));
3906
3907 if isfield(handles, 'plot_popup_value' )
3908     % handles.firing_popup_value exists - you may access it
3909 else
3910     % handles.firing_popup_value does not exist
3911     errordlg('You must choose a Waverform Type!','Invalid Input','modal')
3912     uicontrol(hObject)
3913     return
3914 end
3915
3916 if isnan(rep_amp_ratio_input) || rep_amp_ratio_input == 0 || rem(
         rep_amp_ratio_input , 0.1) ~= 0
3917     errordlg('You must enter a numeric value bigger than zero','Invalid Input','
         modal')
3918     uicontrol(hObject)
3919     return
3920 elseif rep_amp_ratio_input == 0 && handles.repetition == 0
3921     wd = warndlg('There will be no Repetition Cycle','Input','modal')
3922     waitfor(wd);
3923     uicontrol(hObject)
3924     return
3925 elseif handles.repetition == 0
3926     ed = errordlg('There is no Repetition Cycle','Input','modal')
3927     waitfor(ed);
3928     handles.interstimulus_time = 0;
3929     handles.n_repetitions = 1;
```

```
3930      value = handles.plot_popup_value;
3931      string = handles.plot_popup_string;
3932      switch string{value}
3933          case 'Bi-level Lilly' % User selects bilevel Lilly
3934              handles.current_data_I = handles.bi_level;
3935              handles.stimulation_data_I = handles.bi_level;
3936          case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
3937              handles.current_data_I = handles.quasi_trap;
3938              handles.stimulation_data_I = handles.quasi_trap;
3939          case 'Tri-Exponential' % User selects tri-exponential
3940              handles.current_data_I = handles.tri_exp;
3941              handles.stimulation_data_I = handles.tri_exp;
3942          case 'Bursting' % User selects bursting strategies
3943              handles.current_data_I = handles.bursting;
3944              handles.stimulation_data_I = handles.bursting;
3945          case 'Rectangular' % User selects pulse
3946              handles.current_data_I = handles.pulse;
3947              handles.stimulation_data_I = handles.pulse;
3948          case 'Load Waveform'
3949              handles.current_data_I = handles.current_data_I;
3950              handles.stimulation_data_I = handles.current_data_I;
3951      end
3952      return
3953  else
3954      handles.rep_amp_ratio = rep_amp_ratio_input;
3955      value = handles.plot_popup_value;
3956      string = handles.plot_popup_string;
3957      handles.reps_max = 0;
3958      n_reps = 1;
3959      switch string{value}
3960          case 'Bi-level Lilly' % User selects bilevel Lilly
3961              handles.current_data_I = handles.bi_level;
3962              handles.stimulation_data_I = handles.bi_level;
3963              for n_reps = 1:handles.n_repetitions
3964                  if (handles.amplitude_max ~=0 && (handles.amplitude_max <=
      handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
      amplitude_max <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps+1)) )
      && handles.reps_max == 0)...
3965                      || (handles.amplitude_max ==0 && (handles.DAC_amplitude_limit
       <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
      DAC_amplitude_limit <= handles.amplitude_2*((handles.rep_amp_ratio).^(n_reps
      +1)) ) && handles.reps_max == 0)
3966                  handles.reps_max = n_reps;
3967                  end
3968                  if handles.reps_max ~=0
3969                      handles.current_data_I = horzcat(handles.current_data_I,zeros
      (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
      rep_amp_ratio).^(handles.reps_max)).*handles.bi_level);
```

```
3970                 else
3971                     handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.bi_level);
3972                 end
3973             end
3974         case 'Quasi-Trapezoidal' % User selects quasi-trapezoidal
3975             handles.current_data_I = handles.quasi_trap;
3976             handles.stimulation_data_I = handles.quasi_trap;
3977             for n_reps = 1:handles.n_repetitions
3978                 if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
3979                         || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
3980                     handles.reps_max = n_reps;
3981                 end
3982                 if handles.reps_max ~=0
3983                     handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.quasi_trap);
3984                 else
3985                     handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.quasi_trap);
3986                 end
3987             end
3988         case 'Tri-Exponential' % User selects tri-exponential
3989             handles.current_data_I = handles.tri_exp;
3990             handles.stimulation_data_I = handles.tri_exp;
3991             for n_reps = 1:handles.n_repetitions
3992                 if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
3993                         || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
3994                     handles.reps_max = n_reps;
3995                 end
3996                 if handles.reps_max ~=0
3997                     handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.tri_exp);
3998                 else
3999                     handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.tri_exp);
```

```
4000                end
4001            end
4002        case 'Bursting' % User selects bursting strategies
4003            handles.current_data_I = handles.bursting;
4004            handles.stimulation_data_I = handles.bursting;
4005            for n_reps = 1:handles.n_repetitions
4006                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
4007                    || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
4008                    handles.reps_max = n_reps;
4009                end
4010                if handles.reps_max ~=0
4011                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.bursting);
4012                else
4013                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.bursting);
4014                end
4015            end
4016        case 'Rectangular' % User selects pulse
4017            handles.current_data_I = handles.pulse;
4018            handles.stimulation_data_I = handles.pulse;
4019            for n_reps = 1:handles.n_repetitions
4020                if (handles.amplitude_max ~=0 && handles.amplitude_max <= handles
    .amplitude_1*((handles.rep_amp_ratio).^(n_reps+1)) && handles.reps_max == 0)
    ...
4021                    || (handles.amplitude_max ==0 && handles.
    DAC_amplitude_limit <= handles.amplitude_1*((handles.rep_amp_ratio).^(n_reps
    +1)) && handles.reps_max == 0)
4022                    handles.reps_max = n_reps;
4023                end
4024                if handles.reps_max ~=0
4025                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(handles.reps_max)).*handles.pulse);
4026                else
4027                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.pulse);
4028                end
4029            end
4030        case 'Load Waveform'
4031            handles.current_data_I = handles.current_data_I;
```

```
4032            handles.stimulation_data_I = handles.current_data_I;
4033            for n_reps = 1:handles.n_repetitions
4034                if (handles.amplitude_max ~=0 && (handles.amplitude_max <= max(
    handles.stimulation_data_I)*((handles.rep_amp_ratio).^(n_reps+1)) || handles.
    amplitude_max <= abs(min(handles.stimulation_data_I))*((handles.rep_amp_ratio
    ).^(n_reps+1))) && handles.reps_max == 0)...
4035                        || (handles.amplitude_max ==0 && (handles.
    DAC_amplitude_limit <= max(handles.stimulation_data_I)*((handles.
    rep_amp_ratio).^(n_reps+1)) || handles.DAC_amplitude_limit <= abs(min(handles
    .stimulation_data_I))*((handles.rep_amp_ratio).^(n_reps+1))) && handles.
    reps_max == 0)
4036                    handles.reps_max = n_reps;
4037                end
4038                if handles.reps_max ~=0
4039                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.stimulation_data_I);
4040                else
4041                    handles.current_data_I = horzcat(handles.current_data_I,zeros
    (1,(handles.interstimulus_time*handles.time_resolution)),((handles.
    rep_amp_ratio).^(n_reps)).*handles.stimulation_data_I);
4042                end
4043            end
4044    end
4045 end
4046
4047 guidata(hObject, handles);
4048
4049
4050
4051
4052
4053 % --- Executes during object creation, after setting all properties.
4054 function rep_amp_ratio_edit_CreateFcn(hObject, eventdata, handles)
4055 % hObject    handle to rep_amp_ratio_edit (see GCBO)
4056 % eventdata  reserved - to be defined in a future version of MATLAB
4057 % handles    empty - handles not created until after all CreateFcns called
4058
4059 % Hint: edit controls usually have a white background on Windows.
4060 %       See ISPC and COMPUTER.
4061 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
4062     set(hObject,'BackgroundColor','white');
4063 end
4064
4065
4066
4067 function time_lag_edit_Callback(hObject, eventdata, handles)
```

```
4068 % hObject    handle to time_lag_edit (see GCBO)
4069 % eventdata  reserved - to be defined in a future version of MATLAB
4070 % handles    structure with handles and user data (see GUIDATA)
4071
4072 % Hints: get(hObject,'String') returns contents of time_lag_edit as text
4073 %        str2double(get(hObject,'String')) returns contents of time_lag_edit as a
          double
4074
4075 time_lag_input = str2double(get(hObject,'string'));
4076 wd = warndlg('This value applies for all channels. Always.','Safe Range');
4077 waitfor(wd);
4078
4079 if isnan(time_lag_input) || time_lag_input == 0
4080     errordlg('You must enter a numeric value bigger than zero','Invalid Input','
          modal')
4081     uicontrol(hObject)
4082     return
4083 else
4084     handles.time_lag = time_lag_input;
4085 end
4086
4087 guidata(hObject, handles);
4088
4089
4090
4091 % --- Executes during object creation, after setting all properties.
4092 function time_lag_edit_CreateFcn(hObject, eventdata, handles)
4093 % hObject    handle to time_lag_edit (see GCBO)
4094 % eventdata  reserved - to be defined in a future version of MATLAB
4095 % handles    empty - handles not created until after all CreateFcns called
4096
4097 % Hint: edit controls usually have a white background on Windows.
4098 %       See ISPC and COMPUTER.
4099 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
          defaultUicontrolBackgroundColor'))
4100     set(hObject,'BackgroundColor','white');
4101 end
4102
4103
4104 % --- Executes on selection change in channel_popup.
4105 function channel_popup_Callback(hObject, eventdata, handles)
4106 % hObject    handle to channel_popup (see GCBO)
4107 % eventdata  reserved - to be defined in a future version of MATLAB
4108 % handles    structure with handles and user data (see GUIDATA)
4109
4110 % Hints: contents = cellstr(get(hObject,'String')) returns channel_popup contents
          as cell array
4111 %        contents{get(hObject,'Value')} returns selected item from channel_popup
```

```
4112
4113 val = get(handles.channel_popup, 'Value');
4114 handles.channel_popup_value = val;
4115 str = get(hObject, 'String');
4116 handles.channel_popup_string = str;
4117 switch str{val}
4118     case '1'
4119         handles.channel_no = 1;
4120     case '2'
4121         handles.channel_no = 2;
4122     case '3'
4123         handles.channel_no = 3;
4124     case '4'
4125         handles.channel_no = 4;
4126 end
4127
4128 guidata(hObject, handles);
4129
4130
4131 % --- Executes during object creation, after setting all properties.
4132 function channel_popup_CreateFcn(hObject, eventdata, handles)
4133 % hObject    handle to channel_popup (see GCBO)
4134 % eventdata  reserved - to be defined in a future version of MATLAB
4135 % handles    empty - handles not created until after all CreateFcns called
4136
4137 % Hint: popupmenu controls usually have a white background on Windows.
4138 %       See ISPC and COMPUTER.
4139 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
4140     set(hObject,'BackgroundColor','white');
4141 end
4142
4143
4144 % --- Executes on selection change in firing_popup.
4145 function firing_popup_Callback(hObject, eventdata, handles)
4146 % hObject    handle to firing_popup (see GCBO)
4147 % eventdata  reserved - to be defined in a future version of MATLAB
4148 % handles    structure with handles and user data (see GUIDATA)
4149
4150 % Hints: contents = cellstr(get(hObject,'String')) returns firing_popup contents
        as cell array
4151 %        contents{get(hObject,'Value')} returns selected item from firing_popup
4152
4153 val = get(handles.firing_popup, 'Value');
4154 handles.firing_popup_value = val;
4155 str = get(hObject, 'String');
4156 handles.firing_popup_string = str;
4157 switch str{val}
```

```
4158      case 'alone'
4159          handles.firing_type = 1;
4160          set(handles.time_lag_edit, 'enable', 'off')
4161      case 'synchronous'
4162          handles.firing_type = 2;
4163          set(handles.time_lag_edit, 'enable', 'off')
4164      case 'asynchronous'
4165          handles.firing_type = 3;
4166          set(handles.time_lag_edit, 'enable', 'on','String', '1')
4167
4168  end
4169
4170  guidata(hObject, handles);
4171
4172
4173  % --- Executes during object creation, after setting all properties.
4174  function firing_popup_CreateFcn(hObject, eventdata, handles)
4175  % hObject    handle to firing_popup (see GCBO)
4176  % eventdata  reserved - to be defined in a future version of MATLAB
4177  % handles    empty - handles not created until after all CreateFcns called
4178
4179  % Hint: popupmenu controls usually have a white background on Windows.
4180  %       See ISPC and COMPUTER.
4181  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
         defaultUicontrolBackgroundColor'))
4182      set(hObject,'BackgroundColor','white');
4183  end
4184
4185
4186  % --- Executes on selection change in COM_popup.
4187  function COM_popup_Callback(hObject, eventdata, handles)
4188  % hObject    handle to COM_popup (see GCBO)
4189  % eventdata  reserved - to be defined in a future version of MATLAB
4190  % handles    structure with handles and user data (see GUIDATA)
4191
4192  % Hints: contents = cellstr(get(hObject,'String')) returns COM_popup contents as
         cell array
4193  %        contents{get(hObject,'Value')} returns selected item from COM_popup
4194
4195  serialInfo = instrhwinfo('serial');
4196
4197  if isempty(serialInfo.SerialPorts) == 1
4198      errordlg('There are no available COM ports','Invalid Input','modal')
4199      uicontrol(hObject)
4200      return
4201  else
4202      n_coms = 1;
4203      for n_coms = 1:length(serialInfo.SerialPorts)
```

```
4204            string_list(n_coms+1) = serialInfo.SerialPorts(n_coms);
4205        end
4206        set(handles.COM_popup,'String',string_list);
4207    end
4208
4209    val = get(handles.COM_popup, 'Value');
4210    handles.COM_popup_value = val;
4211    str = get(hObject, 'String');
4212    handles.COM_popup_string = str;
4213
4214    guidata(hObject, handles);
4215
4216
4217
4218
4219
4220    % --- Executes during object creation, after setting all properties.
4221    function COM_popup_CreateFcn(hObject, eventdata, handles)
4222    % hObject    handle to COM_popup (see GCBO)
4223    % eventdata  reserved - to be defined in a future version of MATLAB
4224    % handles    empty - handles not created until after all CreateFcns called
4225
4226    % Hint: popupmenu controls usually have a white background on Windows.
4227    %       See ISPC and COMPUTER.
4228    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
4229        set(hObject,'BackgroundColor','white');
4230    end
4231
4232
4233    % --- Executes on button press in apply_pushbutton.
4234    function apply_pushbutton_Callback(hObject, eventdata, handles)
4235    % hObject    handle to apply_pushbutton (see GCBO)
4236    % eventdata  reserved - to be defined in a future version of MATLAB
4237    % handles    structure with handles and user data (see GUIDATA)
4238
4239    if handles.old_channel == 0
4240        errordlg('Nothing uploaded to apply','Invalid Input','modal')
4241        uicontrol(hObject)
4242    else
4243
4244        if isfield(handles, 'COM_popup_value' )
4245            % handles.firing_popup_value exists - you may access it
4246        else
4247            % handles.firing_popup_value does not exist
4248            errordlg('No COM port selected!','Invalid Input','modal')
4249            uicontrol(hObject)
4250
```

```
4251          serialInfo = instrhwinfo('serial');
4252          if isempty(serialInfo.SerialPorts) == 1
4253              errordlg('There are no available COM ports','Invalid Input','modal')
4254              uicontrol(hObject)
4255              return
4256          else
4257              n_coms = 1;
4258              for n_coms = 1:length(serialInfo.SerialPorts)
4259                  string_list(n_coms+1) = serialInfo.SerialPorts(n_coms);
4260              end
4261              set(handles.COM_popup,'String',string_list);
4262          end
4263
4264          return
4265      end
4266
4267
4268      % Attempt Communication with Microcontroller
4269      priorPorts = instrfind; % finds any existing Serial Ports in MATLAB
4270      delete(priorPorts) % and deletes them
4271
4272      s = serial(handles.COM_popup_string{handles.COM_popup_value});
4273
4274      % s = serial('COM6'); %assigns the object s to serial port
4275
4276      set(s, 'InputBufferSize', 1); %128 originally %number of bytes in inout
         buffer
4277      set(s, 'FlowControl', 'none');
4278      set(s, 'BaudRate', 38400);
4279      set(s, 'Parity', 'none');
4280      set(s, 'DataBits', 8);
4281      set(s, 'StopBit', 1);
4282      set(s, 'Timeout',100);
4283      %clc;
4284
4285      try
4286          fopen(s);
4287      catch err
4288          errordlg('Something went wrong with the connection. Maybe another program
          using the device? If not, try re-connect the device',...
4289              'Invalid Input','modal')
4290          uicontrol(hObject)
4291          return
4292      end
4293
4294
4295      final_answer = questdlg('Are you sure you want to apply stimulation?','
         Stimulation','Apply','Cancel','Cancel');
```

```
4296      switch final_answer
4297          case 'Apply'
4298              fprintf(s, '%d\r', -14)
4299              pause(0.02);
4300              fprintf(s, '%d\r', -5)
4301              pause(0.02);
4302              handles.number_sent_channels = 1;
4303              handles.old_channel = 0;
4304              handles.old_firing = 0;
4305          case 'Cancel'
4306      end
4307
4308      fclose(s);
4309      delete(s)
4310      clear s;
4311  end
4312  guidata(hObject, handles);
4313
4314
4315
4316  % --- Executes on button press in reset_pushbutton.
4317  function reset_pushbutton_Callback(hObject, eventdata, handles)
4318  % hObject    handle to reset_pushbutton (see GCBO)
4319  % eventdata  reserved - to be defined in a future version of MATLAB
4320  % handles    structure with handles and user data (see GUIDATA)
4321
4322  if isfield(handles, 'COM_popup_value' )
4323      % handles.firing_popup_value exists - you may access it
4324  else
4325      % handles.firing_popup_value does not exist
4326      errordlg('No COM port selected!','Invalid Input','modal')
4327      uicontrol(hObject)
4328
4329      serialInfo = instrhwinfo('serial');
4330      if isempty(serialInfo.SerialPorts) == 1
4331          errordlg('There are no available COM ports','Invalid Input','modal')
4332          uicontrol(hObject)
4333          return
4334      else
4335          n_coms = 1;
4336          for n_coms = 1:length(serialInfo.SerialPorts)
4337              string_list(n_coms+1) = serialInfo.SerialPorts(n_coms);
4338          end
4339          set(handles.COM_popup,'String',string_list);
4340      end
4341
4342      return
4343  end
```

```
4344
4345
4346 % Attempt Communication with Microcontroller
4347 priorPorts = instrfind; % finds any existing Serial Ports in MATLAB
4348 delete(priorPorts) % and deletes them
4349
4350 s = serial(handles.COM_popup_string{handles.COM_popup_value});
4351
4352 % s = serial('COM6'); %assigns the object s to serial port
4353
4354 set(s, 'InputBufferSize', 1); %128 originally %number of bytes in inout buffer
4355 set(s, 'FlowControl', 'none');
4356 set(s, 'BaudRate', 38400);
4357 set(s, 'Parity', 'none');
4358 set(s, 'DataBits', 8);
4359 set(s, 'StopBit', 1);
4360 set(s, 'Timeout',100);
4361 %clc;
4362
4363 try
4364     fopen(s);
4365 catch err
4366     errordlg('Something went wrong with the connection. Maybe another program
     using the device? If not, try re-connect the device',...
4367         'Invalid Input','modal')
4368     uicontrol(hObject)
4369     return
4370 end
4371
4372
4373 final_answer = questdlg('Are you sure you want to reset ALL the settings already
     uploaded?','Stimulation','Reset','Cancel','Cancel');
4374 switch final_answer
4375     case 'Reset'
4376         fprintf(s, '%d\r', -14)  %because the upload cycle only ends when user '
     Apply's or 'Reset's
4377         pause(0.02);
4378         fprintf(s, '%d\r', -6)
4379         pause(0.02);
4380         handles.number_sent_channels = 1;
4381         handles.old_channel = 0;
4382         handles.old_firing = 0;
4383     case 'Cancel'
4384 end
4385
4386 fclose(s);
4387 delete(s)
4388 clear s;
```

```
4389
4390 guidata(hObject, handles);
4391
4392
4393 function electrode_imped_edit_Callback(hObject, eventdata, handles)
4394 % hObject    handle to electrode_imped_edit (see GCBO)
4395 % eventdata  reserved - to be defined in a future version of MATLAB
4396 % handles    structure with handles and user data (see GUIDATA)
4397
4398 % Hints: get(hObject,'String') returns contents of electrode_imped_edit as text
4399 %        str2double(get(hObject,'String')) returns contents of
         electrode_imped_edit as a double
4400 electrode_imped_input = str2double(get(hObject,'string'));
4401 if isnan(electrode_imped_input)
4402     wd = warndlg('No Amplitude Safe Range Defined!','Safe Range');
4403     waitfor(wd);
4404     handles.amplitude_max = 0;
4405     set(handles.amplitude_maximum_edit, 'enable', 'off','String', ' ')
4406 else
4407     handles.resistance = electrode_imped_input;
4408     handles.amplitude_max = (handles.opamp_max_output_voltage * handles.R_set)/(
         handles.resistance + handles.R_set + (2*handles.R_internal));
4409     set(handles.amplitude_maximum_edit, 'enable', 'off','String', handles.
         amplitude_max)
4410 end
4411
4412 guidata(hObject, handles);
4413
4414
4415 % --- Executes during object creation, after setting all properties.
4416 function electrode_imped_edit_CreateFcn(hObject, eventdata, handles)
4417 % hObject    handle to electrode_imped_edit (see GCBO)
4418 % eventdata  reserved - to be defined in a future version of MATLAB
4419 % handles    empty - handles not created until after all CreateFcns called
4420
4421 % Hint: edit controls usually have a white background on Windows.
4422 %       See ISPC and COMPUTER.
4423 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
         defaultUicontrolBackgroundColor'))
4424     set(hObject,'BackgroundColor','white');
4425 end
4426
4427
4428
4429 function amplitude_maximum_edit_Callback(hObject, eventdata, handles)
4430 % hObject    handle to amplitude_maximum_edit (see GCBO)
4431 % eventdata  reserved - to be defined in a future version of MATLAB
4432 % handles    structure with handles and user data (see GUIDATA)
```

```
4433
4434 % Hints: get(hObject,'String') returns contents of amplitude_maximum_edit as text
4435 %        str2double(get(hObject,'String')) returns contents of
         amplitude_maximum_edit as a double
4436
4437
4438 % --- Executes during object creation, after setting all properties.
4439 function amplitude_maximum_edit_CreateFcn(hObject, eventdata, handles)
4440 % hObject    handle to amplitude_maximum_edit (see GCBO)
4441 % eventdata  reserved - to be defined in a future version of MATLAB
4442 % handles    empty - handles not created until after all CreateFcns called
4443
4444 % Hint: edit controls usually have a white background on Windows.
4445 %       See ISPC and COMPUTER.
4446 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
         defaultUicontrolBackgroundColor'))
4447     set(hObject,'BackgroundColor','white');
4448 end
```

Appendix B

Microcontroller Code

B.1 Main Function Code

```
/* ###################################################################
**     Filename    : main.c
**     Project     : Send_Num
**     Processor   : MKL26Z128VLH4
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2014-11-19, 12:12, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file main.c
** @version 01.01
** @brief
**         Main module.
**         This module contains user's application code.
*/
/*!
**  @addtogroup main_module main module documentation
**  @{
*/
/* MODULE main */

```

```
29
30 /*
       ###################################################################################
31 ATTENTION: For this program to run correctly, the WAIT component has to be
       downloaded
32 Please search 'MCUonEclipse' and 'Processor Expert Component *.PEupd Files on
       GitHub'
33 to understando how to add WAIT component
34 ***************************************************************************************
       */
35
36
37 /* Including needed modules to compile this module/procedure */
38 #include "Cpu.h"
39 #include "Events.h"
40 #include "CsIO1.h"
41 #include "IO1.h"
42 #include "WAIT1.h"
43 #include "BLUE.h"
44 #include "IN11.h"
45 #include "GREEN.h"
46 #include "RED.h"
47 #include "IN21.h"
48 #include "IN12.h"
49 #include "IN22.h"
50 #include "IN13.h"
51 #include "IN23.h"
52 #include "IN14.h"
53 #include "IN24.h"
54 #include "SHDN1.h"
55 #include "SHDN2.h"
56 #include "EN.h"
57 /* Including shared modules, which are used for whole project */
58 #include "PE_Types.h"
59 #include "PE_Error.h"
60 #include "PE_Const.h"
61 #include "IO_Map.h"
62
63 /* User includes (#include below this line is not maintained by Processor Expert)
       */
64 #include <stdio.h>
65 #include <stdlib.h>
66
67 /* Constant definition */
68 #define DATA_LENGTH_MAX 930 // Value bigger than this gives memory-related error
69 #define CHANNELS 5 // Array 0 (zero) + 4 channels
70 #define SWITCHOFF 0
```

```
71 #define SWITCHON01 1
72 #define SWITCHON10 2
73
74 /* This definition speeds up these ON/OFF outputs */
75 #define IN11_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN11_MODULE_BASE_ADDRESS,
       IN11_PORT_MASK)
76 #define IN11_Set()     GPIO_PDD_SetPortDataOutputMask(IN11_MODULE_BASE_ADDRESS,
      IN11_PORT_MASK)
77 #define IN21_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN21_MODULE_BASE_ADDRESS,
       IN21_PORT_MASK)
78 #define IN21_Set()     GPIO_PDD_SetPortDataOutputMask(IN21_MODULE_BASE_ADDRESS,
      IN21_PORT_MASK)
79
80 #define IN12_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN12_MODULE_BASE_ADDRESS,
       IN12_PORT_MASK)
81 #define IN12_Set()     GPIO_PDD_SetPortDataOutputMask(IN12_MODULE_BASE_ADDRESS,
      IN12_PORT_MASK)
82 #define IN22_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN22_MODULE_BASE_ADDRESS,
       IN22_PORT_MASK)
83 #define IN22_Set()     GPIO_PDD_SetPortDataOutputMask(IN22_MODULE_BASE_ADDRESS,
      IN22_PORT_MASK)
84
85 #define IN13_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN13_MODULE_BASE_ADDRESS,
       IN13_PORT_MASK)
86 #define IN13_Set()     GPIO_PDD_SetPortDataOutputMask(IN13_MODULE_BASE_ADDRESS,
      IN13_PORT_MASK)
87 #define IN23_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN23_MODULE_BASE_ADDRESS,
       IN23_PORT_MASK)
88 #define IN23_Set()     GPIO_PDD_SetPortDataOutputMask(IN23_MODULE_BASE_ADDRESS,
      IN23_PORT_MASK)
89
90 #define IN14_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN14_MODULE_BASE_ADDRESS,
       IN14_PORT_MASK)
91 #define IN14_Set()     GPIO_PDD_SetPortDataOutputMask(IN14_MODULE_BASE_ADDRESS,
      IN14_PORT_MASK)
92 #define IN24_Clr()     GPIO_PDD_ClearPortDataOutputMask(IN24_MODULE_BASE_ADDRESS,
       IN24_PORT_MASK)
93 #define IN24_Set()     GPIO_PDD_SetPortDataOutputMask(IN24_MODULE_BASE_ADDRESS,
      IN24_PORT_MASK)
94
95
96 void MCU_Init(void)
97 {
98      //SPI0 module initialization
99      SIM_SCGC4 |= SIM_SCGC4_SPI0_MASK;        // Turn on clock to SPI0 module
100     SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK;       // Turn on clock to Port D module
101     PORTC_PCR5 = PORT_PCR_MUX(0x02);         // PTC5 pin is SPI0 CLK line
102     PORTC_PCR6 = PORT_PCR_MUX(0x02);         // PTC6 pin is SPI0 MOSI line
```

```
103      PORTC_PCR7 = PORT_PCR_MUX(0x02);          // PTC7 pin is SPI0 MISO line
104      PORTC_PCR4 = PORT_PCR_MUX(0x02);          // PTC4 pin is configured SSbar
105
106      SPI0_C1 = SPI_C1_SPE_MASK | SPI_C1_MSTR_MASK | SPI_C1_SSOE_MASK;     //
       Enable SPI0 module, master mode, SS low active mode
107      SPI0_C2 = SPI_C2_SPIMODE_MASK | SPI_C2_MODFEN_MASK;
108      SPI0_BR = SPI_BR_SPPR(0x00) | SPI_BR_SPR(0x00);     // BaudRate = BusClock /
       ((SPPR+1) * 2^(SPR+1)) = 20970000 / ((1+1) * 2^(0+1)) = 5.2425 MHz
109      SPI0_S = SPI_S_SPTEF_MASK;
110
111  }
112
113
114
115  /*lint -save  -e970 Disable MISRA rule (6.3) checking. */
116  int main(void)
117  /*lint -restore Enable MISRA rule (6.3) checking. */
118  {
119
120          /* Write your local variable definition here */
121
122
123          /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!!
       ***/
124          PE_low_level_init();
125          /*** End of Processor Expert internal initialization.
       ***/
126
127          /* SPI initialization */
128          MCU_Init();
129
130          /* Turn ON Power Supplies */
131          SHDN1_SetVal(SHDN1_DeviceData);
132          SHDN2_SetVal(SHDN1_DeviceData);
133          EN_SetVal(EN_DeviceData);
134
135
136          for(;;) {
137
138                  /* Variables initialization */
139
140                  //MATLAB information variables
141                  int value = 0;
142                  int swtvalue = 0; //needed only if switch is intended to stay on
       the all time instead of just a peak
143                  int channel = 0;
144                  int timeLag = 0;
145                  int firingType = 0;
```

```
146                 int repetitonAmpRatio[CHANNELS] =  {0};
147                 int invRatio[CHANNELS] =  {0}; //needed because transmission of
        float values through UART leads to errors
148                 int interstimulusTime[CHANNELS] =  {0};  // applicable for Alone
        and Asynchronous Fire
149                 int nRepetitions[CHANNELS] =  {0};      // in here channels is
        possible because one can terminate earlier than the others
150                 int RepsMax[CHANNELS] = {0};
151                 int interstimulusTimeSynch = 0; // applicable only for
        Synchronous Fire
152
153                 // Counters
154                 int pwr = 0;
155                 int n = 0;
156                 int i = 0;
157                 int rep = 0;
158                 int rep1 = 0;
159                 int rep2 = 0;
160                 int rep3 = 0;
161                 int rep4 = 0;
162                 int nRepts = 0;
163
164                 // Output Arrays
165                 uint8_t data = {0};
166                 uint8_t addressData = {0};
167
168                 // One Pulse Data Arrays - they are defined as 16-bit but only
        use up to 12-bit numbers because that is the DAC maximum
169                 uint16_t current1[DATA_LENGTH_MAX] = {0}; //Due to memory
        overload, it has to be like this, can't be a [channel][DATA_LENGTH_MAX]
170                 uint16_t current2[DATA_LENGTH_MAX] = {0};
171                 uint16_t current3[DATA_LENGTH_MAX] = {0};
172                 uint16_t current4[DATA_LENGTH_MAX] = {0};
173                 uint16_t controller1[DATA_LENGTH_MAX] = {0};
174                 uint16_t controller2[DATA_LENGTH_MAX] = {0};
175                 uint16_t controller3[DATA_LENGTH_MAX] = {0};
176                 uint16_t controller4[DATA_LENGTH_MAX] = {0};
177
178                 // LED  OFF and Switch Controllers LOW
179                 BLUE_SetVal(BLUE_DeviceData); // turn OFF blue led
180                 RED_SetVal(RED_DeviceData); // turn OFF red led
181                 GREEN_ClrVal(GREEN_DeviceData); // turn ON green led - this is
        the only one ON, to let the user know that it is working
182                 IN11_Clr();     // turn LOW switch 1 from channel 1
183                 IN21_Clr();     // turn LOW switch 2 from channel 1
184                 IN12_Clr();     // turn LOW switch 1 from channel 2
185                 IN22_Clr();     // turn LOW switch 2 from channel 2
186                 IN13_Clr();     // turn LOW switch 1 from channel 3
```

```
187                 IN23_Clr();     // turn LOW switch 2 from channel 3
188                 IN14_Clr();     // turn LOW switch 1 from channel 4
189                 IN24_Clr();     // turn LOW     switch 2 from channel 4
190
191                 // To Power Down the DAC Output
192                 data = (0x00 & 0xFF);   // least significant byte (8-bit from the
        right)
193                 addressData = (0x00 >> 8) + 0x70;       // shift to get the
        remaining 4-bits of the 12-bit
194                 while(!(SPI0_S)); // A wait cycle while SPI information is not
        sent
195                 SPI0_DH=addressData; //DAC select A[1:0], State select OP[1:0],
        Data bits D[11:8]
196                 SPI0_DL=data;   // Data bits D[7:0]
197
198
199                 /* Cycle to receive the information and data values from MATLAB
        */
200                 /*** Communication protocol between MATLAB and MCU is based in
        negative values because all important information and data is sent in
        positive integers ***/
201                 for(;;){
202
203                         // Reading from UART. Blocks cycle until a value is
        received
204                         char line[20];
205                         fgets(line, sizeof(line), stdin);
206                         sscanf(line, "%d", &value);
207
208                         // When a value is received is, green and blue leds are
        OFF, red led turns ON
209                         GREEN_SetVal(GREEN_DeviceData);
210                         BLUE_SetVal(BLUE_DeviceData);
211                         RED_ClrVal(RED_DeviceData);
212
213                         // The MATLAB 'pre-warns' the MCU of what is the
        information that is going to send.
214                         if (value == -7){  // MATLAB is going to send information
         on the Channel and values from one pulser
215
216                                 char line[20];
217                                 fgets(line, sizeof(line), stdin);
218                                 sscanf(line, "%d", &value);
219
220                                 channel = value;
221
222                                 // If are data values regarding the channel 1
223                                 if (channel == 1) {
```

```
224                                     n = 0;
225                                     while (1){
226                                             char line[20];
227                                             fgets(line, sizeof(line), stdin);
228                                             sscanf(line, "%d", &value);
229
230                                             if (n > DATA_LENGTH_MAX){ // If, somehow, the length of one pulse is bigger than the possible
231                                                     break;
                        // the cycle breaks and a restart of the system will occur
232                                             }
233                                             else if (value == -1){  //If in the pulse there will be a zero phase (interphasic delay)
234
235                                                     swtvalue = SWITCHOFF; // the value that means switch off (0) is saved
236
237                                             }
238                                             else if (value == -2){  //If in the pulse there is a variation to a positive phase
239
240                                                     swtvalue = SWITCHON01; // the value that means the respective switch on (1) is saved
241
242                                             }
243                                             else if (value == -3){  //If in the pulse there is a variation to a negative phase
244
245                                                     swtvalue = SWITCHON10; // the value that means the respective switch on (2) is saved
246
247                                             }
248                                             else if (value == -4){  //If it is the end of data on one pulse
249
250                                                     break;  // the cycle breaks and a restart of the system will occur
251                                             }
252
253                                             else {  //If it is not any information on variation or to break cycle then it is data of the pulse to store
254                                                     controller1[n] = swtvalue; // Stores the 'configuration' of the switches in the same place where the respective phase starts
255                                                     current1[n] = value; // Stores the value of the pulse
256                                                     n++;
```

```
257
258                                             }
259                                     }
260
261                             }
262
263
264
265                             if (channel == 2) {
266                                     n = 0;
267                                     while (1){
268                                             char line[20];
269                                             fgets(line, sizeof(line), stdin);
270                                             sscanf(line, "%d", &value);
271
272                                             if (n > DATA_LENGTH_MAX){
273                                                     break;
274                                             }
275                                             else if (value == -1){  //
    shorting phase - interphase
276
277                                                     swtvalue = SWITCHOFF;
278
279                                             }
280                                             else if (value == -2){  //
    positive phase
281
282                                                     swtvalue = SWITCHON01;
283
284                                             }
285                                             else if (value == -3){  //
    negative phase
286
287                                                     swtvalue = SWITCHON10;
288
289                                             }
290                                             else if (value == -4){
291
292                                                     break;
293                                             }
294
295                                             else {
296                                                     controller2[n] = swtvalue
    ;
297                                                     current2[n] = value;
298                                                     n++;
299
300                                             }
```

```
301                                         }
302
303                                 }
304
305
306
307
308                                 if (channel == 3) {
309                                         n = 0;
310                                         while (1){
311                                                 char line[20];
312                                                 fgets(line, sizeof(line), stdin);
313                                                 sscanf(line, "%d", &value);
314
315                                                 if (n > DATA_LENGTH_MAX){
316                                                         break;
317                                                 }
318                                                 else if (value == -1){  //
    shorting phase - interphase
319
320                                                         swtvalue = SWITCHOFF;
321
322                                                 }
323                                                 else if (value == -2){  //
    positive phase
324
325                                                         swtvalue = SWITCHON01;
326
327                                                 }
328                                                 else if (value == -3){  //
    negative phase
329
330                                                         swtvalue = SWITCHON10;
331
332                                                 }
333                                                 else if (value == -4){
334
335                                                         break;
336                                                 }
337
338                                                 else {
339                                                         controller3[n] = swtvalue
    ;
340                                                         current3[n] = value;
341                                                         n++;
342
343                                                 }
344                                         }
```

```
345
346                                 }
347
348
349
350                                 if (channel == 4) {
351                                         n = 0;
352                                         while (1){
353                                                 char line[20];
354                                                 fgets(line, sizeof(line), stdin);
355                                                 sscanf(line, "%d", &value);
356
357                                                 if (n > DATA_LENGTH_MAX){
358                                                         break;
359                                                 }
360                                                 else if (value == -1){  //
        shorting phase - interphase
361
362                                                         swtvalue = SWITCHOFF;
363
364                                                 }
365                                                 else if (value == -2){  //
        positive phase
366
367                                                         swtvalue = SWITCHON01;
368
369                                                 }
370                                                 else if (value == -3){  //
        negative phase
371
372                                                         swtvalue = SWITCHON10;
373
374                                                 }
375                                                 else if (value == -4){
376
377                                                         break;
378                                                 }
379
380                                                 else {
381                                                         controller4[n] = swtvalue
        ;
382                                                         current4[n] = value;
383                                                         n++;
384
385                                                 }
386                                         }
387
388                                 }
```

```
389
390                         }
391
392                         else if (value == -10){ // MATLAB is going to send
    information on the Repetition Rate
393
394                                 char line[20];
395                                 fgets(line, sizeof(line), stdin);
396                                 sscanf(line, "%d", &value);
397
398                                 nRepetitions[channel] = value; // Saves the
    number of repetitions of the respective channel
399
400
401                                 char line2[20];
402                                 fgets(line2, sizeof(line2), stdin);
403                                 sscanf(line2, "%d", &value);
404
405                                 if (firingType == 2){
406                                         interstimulusTimeSynch = value; // Saves
    the interstimulus time of all channels in synchronised firing
407                                 }
408                                 else {
409                                         interstimulusTime[channel] = value;
    // Saves the interstimulus time of each channel in alone and asynchronised
    firing
410                                 }
411
412
413                                 char line3[20];
414                                 fgets(line3, sizeof(line3), stdin);
415                                 sscanf(line3, "%d", &value);
416
417                                 RepsMax[channel] = value;
418
419                         }
420                         else if (value == -11){ // MATLAB is going to send
    information on the Repetition Amplitude Ratio
421
422                                 char line[20];
423                                 fgets(line, sizeof(line), stdin);
424                                 sscanf(line, "%d", &value);
425
426                                 if (value == -15){      // If the Repetition
    Amplitude Ratio is between 1 and 0
427                                         invRatio[channel] = -1;
428                                 }
```

```
429                             else if (value == -16){ // If the Repetition
    Amplitude Ratio is higher than 1
430                                     invRatio[channel] = 1;
431                             }
432
433                             char line2[20];
434                             fgets(line2, sizeof(line2), stdin);
435                             sscanf(line2, "%d", &value);
436
437                             repetitonAmpRatio[channel] = value; // Saves the
    Repetition Amplitude Ratio of each channel
438                     }
439                     else if (value == -12){ // MATLAB is going to send
    information on Time Lag
440
441                             char line[20];
442                             fgets(line, sizeof(line), stdin);
443                             sscanf(line, "%d", &value);
444
445                             timeLag = value; // Saves the Time Lag between
    all channel in asynchronous firing
446
447                             RED_SetVal(RED_DeviceData);     // turn off red
    LED
448                             BLUE_ClrVal(BLUE_DeviceData); // turn on blue LED
     - this is located here because it is the last information on a channel sent
    by MATLAB
449
450                     }
451                     else if (value == -13){ // MATLAB is going to send
    information on the Firing Type
452
453                             char line[20];
454                             fgets(line, sizeof(line), stdin);
455                             sscanf(line, "%d", &value);
456
457                             firingType = value; // Saves which Firing Type is
     going to be used
458                     }
459                     else if (value == -14 || n > DATA_LENGTH_MAX){  // User
    in MALTAB has finished programming the channels or wants to reset all
    information already uploaded
460
461                             RED_SetVal(RED_DeviceData); // turn off red LED
462                             BLUE_SetVal(BLUE_DeviceData); // turn off blue
    LED
463                             break; // terminates all communication cycle
464                     }
```

```
465
466                 }
467
468
469                 /// THIS PART STILL HAVE CHANGES TO BE MADE. IN FIRING TYPE = 1,
    CHANNEL 1 AND 2 HAVE DIFFERENT OUTPUT FROM 3 AND 4 ///
470                 // This is due to time testing being made in order to minimize
    delay between the output of the original sample and the output of the sample
    multplied by amplitude repetition ratio //
471
472
473                 /* Code to use the information and data values previously saved
    to obtain a stimulation according to user preferences  */
474                 char line[20];
475                 fgets(line, sizeof(line), stdin);
476                 sscanf(line, "%d", &value);
477
478
479                 if (n > (DATA_LENGTH_MAX+1) || value == -6){  // If amount of
    data is incorrect (due to some transfer error) or user wants to reset all
    informantion
480                         // skips everything and go to the beginning (to
    communication with MATLAB)
481                 }
482                 else if (value == -5) { // If user wants to apply the stimulation
     uploaded
483                         RED_ClrVal(RED_DeviceData);     // turn off red LED
484
485                         // If user chose an Alone Firing
486                         if (firingType == 1){
487                                 nRepts = 0;
488                                 pwr = 0;
489
490                                 // One stimulation pulse (stored in MCU) is
    repeated for the number of repetitions intended in only one channel
491                                 while (nRepts < nRepetitions[channel]) {
492                                 i = 0;
493                                 rep = 0;
494
495                                 // If user chose an Channel 1 for Firing
496                                 if (channel == 1){
497                                         for (i = 0; i <= (n-1); i++) { // For the
     length of the pulse uploaded
498
499                                                 if(controller1[i] == SWITCHOFF &&
     rep!=0){ // If it is a zero phase and it was a variation to zero (this way
    avoiding entering in this condition with every SWITCHOFF)
```

```
500                                                     IN11_Clr(); // Switch 1
    to LOW
501                                                     IN21_Clr(); // Switch 2
    to LOW
502                                                     rep = 0; // just to avoid
     entering in this condition with every SWITCHOFF
503                                                 }
504                                                 else if(controller1[i] ==
    SWITCHON01 && rep!=1){ //positive phase
505                                                     IN11_Set(); // Switch 1
    to UP
506                                                     IN21_Clr(); // Switch 2
    to LOW
507                                                     rep = 1; // just to avoid
     entering in this condition with every SWITCHON01
508                                                 }
509                                                 else if(controller1[i] ==
    SWITCHON10 && rep!=2){ //negative phase
510                                                     IN11_Clr(); // Switch 1
    to LOW
511                                                     IN21_Set(); // Switch 2
    to UP
512                                                     rep = 2;        // just
    to avoid entering in this condition with every SWITCHON10
513                                                 }
514
515
516                                                 //Condition for application of
    Repetition Amplitude Ratio
517                                                 if (pwr > 0 && invRatio[channel]
    == 1 && nRepts <= RepsMax[channel]) { // if ratio is bigger than 1
518                                                     current1[i] = current1[i]
     * (repetitonAmpRatio[channel]/10); // the ratio is divided by 10 because in
    MATLAB is multiplied by 10 to allow user one decimal values
519
520                                                 }
521                                                 else if (pwr > 0 && invRatio[
    channel] == -1 && nRepts <= RepsMax[channel]) { //if ratio is between 1 and 0
522                                                     current1[i] = current1[i]
     / (repetitonAmpRatio[channel]/10);
523
524                                                 }
525                                                 else { // this is included so
    that the delay between repetitions wont be so big
526                                                     current1[i] = current1[i]
     * ((repetitonAmpRatio[channel])/ (repetitonAmpRatio[channel]));
527                                                 }
528
```

```
529                                         data = (current1[i] & 0xFF);
    // least significant byte of the data value (8-bit from the right)
530                                         addressData = (current1[i] >> 8) + 0
    x10;    // shift to get the remaining 4-bits of the 12-bit value
531                                             // for the DAC A and when it is
    alone fire need to add DAC:00 and OP:01 to addressData
532
533                                             while(!(SPI0_S));
534                                             SPI0_DH=addressData;  //DAC
    select A[1:0], State select OP[1:0], Data bits D[11:8]
535                                             SPI0_DL=data;     // Data bits D
    [7:0]
536                                     }
537
538                                     IN11_Clr(); // Switch 1 to LOW
539                                     IN21_Clr(); // Switch 2 to LOW
540
541                                     data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
542                                     addressData = (0x00 >> 8) + 0x10;
    // shift to get the remaining 4-bits of the 12-bit
543                                     // for the DAC A and when it is alone
    fire need to add DAC:00 and OP:01 to addressData
544
545                                     while(!(SPI0_S));
546                                     SPI0_DH=addressData;  //DAC select A
    [1:0], State select OP[1:0], Data bits D[11:8]
547                                     SPI0_DL=data;    // Data bits D[7:0]
548
549                                 }
550                                 else if (channel == 2) {
551                                             for (i = 0; i <= (n-1); i++) {
552
553                                                     if(controller2[i] ==
    SWITCHOFF && rep!=0){
554                                                             IN12_Clr();
555                                                             IN22_Clr();
556                                                             rep = 0;
557                                                     }
558                                                     else if(controller2[i] ==
     SWITCHON01 && rep!=1){ //positive phase
559                                                             IN12_Set();
560                                                             IN22_Clr();
561                                                             rep = 1;
562                                                     }
563                                                     else if(controller2[i] ==
     SWITCHON10 && rep!=2){ //negative phase
564                                                             IN12_Clr();
```

```
565                                                             IN22_Set();
566                                                             rep = 2;
567                                                         }
568
569
570
571                                                         if (pwr > 0 && invRatio[
    channel] == 1 && nRepts <= RepsMax[channel]) {
572                                                             current2[i] =
    current2[i] * (repetitonAmpRatio[channel]/10);
573
574                                                         }
575                                                         else if (pwr > 0 &&
    invRatio[channel] == -1 && nRepts <= RepsMax[channel]) {
576                                                             current2[i] =
    current2[i] / (repetitonAmpRatio[channel]/10);
577
578                                                         }
579                                                         else { // this is
    included so that the delay between repetitions wont be so big
580                                                             current2[i] =
    current2[i] * ((repetitonAmpRatio[channel])/ (repetitonAmpRatio[channel]));
581                                                         }
582
583                                                         data = (current2[i] & 0
    xFF);    // least significant byte (8-bit from the right)
584                                                         addressData = (current2[i
    ] >> 8) + 0x50;                // shift to get the remaining 4-bits of the
    12-bit
585                                                         // for the DAC B and when
     it is alone fire need to add DAC:01 and OP:01 to addressData
586
587                                                         while(!(SPI0_S));
588                                                         SPI0_DH=addressData; //
    DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
589                                                         SPI0_DL=data;    // Data
    bits D[7:0]
590                                                 }
591
592                                                 IN12_Clr();
593                                                 IN22_Clr();
594
595                                                 data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
596                                                 addressData = (0x00 >> 8) + 0x50;
            //adds DAC:01 and OP:00
597
598                                                 while(!(SPI0_S));
```

```
599                                             SPI0_DH=addressData;  //DAC
    select A[1:0], State select OP[1:0], Data bits D[11:8]
600                                             SPI0_DL=data;    // Data bits D
    [7:0]
601
602                                     }
603                                 else if (channel == 3) {
604                                             for (i = 0; i <= (n-1); i++) {
605
606                                                     if(controller3[i] ==
    SWITCHOFF && rep!=0){
607                                                             IN13_Clr();
608                                                             IN23_Clr();
609                                                             rep = 0;
610                                                     }
611                                                     else if(controller3[i] ==
     SWITCHON01 && rep!=1){ //positive phase
612                                                             IN13_Set();
613                                                             IN23_Clr();
614                                                             rep = 1;
615                                                     }
616                                                     else if(controller3[i] ==
     SWITCHON10 && rep!=2){ //negative phase
617                                                             IN13_Clr();
618                                                             IN23_Set();
619                                                             rep = 2;
620                                                     }
621
622
623                                                     if (pwr > 0 && invRatio[
    channel] == 1 && nRepts <= RepsMax[channel]) { // if is not the first pulse
    and ratio is bigger than 1 - application of Repetition Amplitude Ratio
624                                                             current3[i] =
    current3[i] * (repetitonAmpRatio[channel]/10); // the ratio is divided by 10
    because in MATLAB is multiplied by 10 to allow user one decimal values
625
626                                                     }
627                                                     else if (pwr > 0 &&
    invRatio[channel] == -1 && nRepts <= RepsMax[channel]) { // if is not the
    first pulse and ratio is between 0 and 1 - application of Repetition
    Amplitude Ratio
628                                                             current3[i] =
    current3[i] / (repetitonAmpRatio[channel]/10);
629
630                                                     }
631                                                     else { // this is
    included so that the delay between repetitions wont be so big
```

```
632                                                         current3[i] =
    current3[i] * ((repetitonAmpRatio[channel]/10)/ (repetitonAmpRatio[channel
    ]/10));
633                                                     }
634
635                                                     data = (current3[i] & 0
    xFF);    // least significant byte (8-bit from the right)
636                                                     addressData = (current3[i
    ] >> 8) + 0x90;        // shift to get the remaining 4-bits of the 12-bit
637                                                     // for the DAC C and when
     it is alone fire need to add DAC:10 and OP:01 to addressData
638
639                                                     while(!(SPI0_S));
640                                                     SPI0_DH=addressData;  //
    DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
641                                                     SPI0_DL=data;    // Data
    bits D[7:0]
642                                             }
643
644                                             IN13_Clr();
645                                             IN23_Clr();
646
647                                             data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
648                                             addressData = (0x00 >> 8) + 0x90;
            //adds DAC:10 and OP:00
649
650                                             while(!(SPI0_S));
651                                             SPI0_DH=addressData;  //DAC
    select A[1:0], State select OP[1:0], Data bits D[11:8]
652                                             SPI0_DL=data;    // Data bits D
    [7:0]
653
654                                     }
655                                 else if (channel == 4) {
656                                             for (i = 0; i <= (n-1); i++) {
657
658                                                     if(controller4[i] ==
    SWITCHOFF && rep!=0){
659                                                             IN14_Clr();
660                                                             IN24_Clr();
661                                                             rep = 0;
662                                                     }
663                                                     else if(controller4[i] ==
     SWITCHON01 && rep!=1){ //positive phase
664                                                             IN14_Set();
665                                                             IN24_Clr();
666                                                             rep = 1;
```

```
667                                                     }
668                                                     else if(controller4[i] ==
     SWITCHON10 && rep!=2){ //negative phase
669                                                             IN14_Clr();
670                                                             IN24_Set();
671                                                             rep = 2;
672                                                     }
673
674
675                                                     if (pwr > 0 && invRatio[
    channel] == 1 && nRepts <= RepsMax[channel]) {
676                                                             current4[i] =
    current4[i] * (repetitonAmpRatio[channel]/10);
677
678                                                     }
679                                                     else if (pwr > 0 &&
    invRatio[channel] == -1 && nRepts <= RepsMax[channel]) {
680                                                             current4[i] =
    current4[i] / (repetitonAmpRatio[channel]/10);
681
682                                                     }
683                                                     else { //included this so
     that the delay between repetitions wont be so big
684                                                             current4[i] =
    current4[i] * ((repetitonAmpRatio[channel]*10)/ (repetitonAmpRatio[channel
    ]*10));
685                                                     }
686
687                                                     data = (current4[i] & 0
    xFF);    // least significant byte (8-bit from the right)
688                                                     addressData = (current4[i
    ] >> 8) + 0xD0;                 // shift to get the remaining 4-bits of the
    12-bit
689                                                     // for the DAC D and when
     it is alone fire need to add DAC:11 and OP:01 to addressData
690
691                                                     while(!(SPI0_S));
692                                                     SPI0_DH=addressData;  //
    DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
693                                                     SPI0_DL=data;   // Data
    bits D[7:0]
694                                             }
695
696                                             IN14_Clr();
697                                             IN24_Clr();
698
699                                             data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
```

```
700                                             addressData = (0x00 >> 8) + 0xD0;
        //adds DAC:11 and OP:01
701
702                                             while(!(SPI0_S));
703                                             SPI0_DH=addressData;  //DAC
    select A[1:0], State select OP[1:0], Data bits D[11:8]
704                                             SPI0_DL=data;    // Data bits D
    [7:0]
705
706                                     }
707
708                             //// WAIT FOR INTERSTIMULUS TIME //////
709                             WAIT1_Waitus(interstimulusTime[channel]);
710                             nRepts++;
711                             pwr++;
712                             }
713                     }
714
715                     // A stimulation pulse is applied in the chosen channels
    at the same time, with the same frequency, and is repeated for the number of
    repetitions intended
716                     else if (firingType == 2){
717                             nRepts = 0;
718                             pwr = 0;
719
720                             while (nRepts < nRepetitions[1] || nRepts <
    nRepetitions[2] || nRepts < nRepetitions[3] || nRepts < nRepetitions[4]) {
721                                     i = 0;
722                                     rep1 = 0;
723                                     rep2 = 0;
724                                     rep3 = 0;
725                                     rep4 = 0;
726                                     for (i = 0; i <= (n-1); i++) {
727
728                                             if (nRepts < nRepetitions[1]){
729
730                                               if(controller1[i] == SWITCHOFF
    && rep1!=0){
731                                                       IN11_Clr();
732                                                       IN21_Clr();
733                                                       rep1 = 0;
734                                               }
735                                               else if(controller1[i] ==
    SWITCHON01 && rep1!=1){ //positive phase
736                                                       IN11_Set();
737                                                       IN21_Clr();
738                                                       rep1 = 1;
739                                               }
```

```
740                                                 else if(controller1[i] ==
    SWITCHON10 && rep1!=2){ //negative phase
741                                                   IN11_Clr();
742                                                   IN21_Set();
743                                                   rep1 = 2;
744                                                 }
745
746
747                                                       if (pwr > 0 && invRatio
    [1] == 1 && nRepts <= RepsMax[1]) {
748                                                             current1[i] =
    current1[i] * (repetitonAmpRatio[1]/10);
749
750                                                       }
751                                                       else if (pwr > 0 &&
    invRatio[1] == -1 && nRepts <= RepsMax[1]) {
752                                                             current1[i] =
    current1[i] / (repetitonAmpRatio[1]/10);
753
754                                                       }
755                                                       else { //included this so
     that the delay between repetitions wont be so big
756                                                             current1[i] =
    current1[i] * ((repetitonAmpRatio[1]*10)/ (repetitonAmpRatio[1]*10));
757                                                       }
758
759                                                 data = (current1[i] & 0xFF);
    // least significant byte (8-bit from the right)
760                                                 addressData = (current1[i] >>
    8);     // shift to get the remaining 4-bits of the 12-bit
761                                                 // for the DAC A and when it is
     synchronous fire no need to add nothing to addressData
762
763                                                 while(!(SPI0_S));
764                                                 SPI0_DH=addressData;  //DAC
    select A[1:0], State select OP[1:0], Data bits D[11:8]
765                                                 SPI0_DL=data;  // Data bits D
    [7:0]
766
767                                               }
768
769
770
771                                               if (nRepts < nRepetitions[2]){
772
773                                                 if(controller2[i] == SWITCHOFF
    && rep2!=0){
774                                                   IN12_Clr();
```

```
775                                                     IN22_Clr();
776                                                     rep2 = 0;
777                                                 }
778                                                 else if(controller2[i] ==
    SWITCHON01 && rep2!=1){ //positive phase
779                                                     IN12_Set();
780                                                     IN22_Clr();
781                                                     rep2 = 1;
782                                                 }
783                                                 else if(controller2[i] ==
    SWITCHON10 && rep2!=2){ //negative phase
784                                                     IN12_Clr();
785                                                     IN22_Set();
786                                                     rep2 = 2;
787                                                 }
788
789                                                     if (pwr > 0 && invRatio
    [2] == 1  && nRepts <= RepsMax[2]) {
790                                                         current2[i] =
    current2[i] * (repetitonAmpRatio[2]/10);
791
792                                                     }
793                                                     else if (pwr > 0 &&
    invRatio[2] == -1 && nRepts <= RepsMax[2]) {
794                                                         current2[i] =
    current2[i] / (repetitonAmpRatio[2]/10);
795
796                                                     }
797                                                     else { //included this so
     that the delay between repetitions wont be so big
798                                                         current2[i] =
    current2[i] * ((repetitonAmpRatio[2]*10) / (repetitonAmpRatio[2]*10));
799                                                     }
800
801                                                 // for the DAC B and when it is
     synchronous fire no need to add nothing to addressData
802                                                 data = (current2[i] & 0xFF);
    // least significant byte (8-bit from the right)
803                                                 addressData = (current2[i] >>
    8) + 0x40;       //adds DAC:01 and OP:00
804
805                                                 while(!(SPI0_S));
806                                                 SPI0_DH=addressData;  //DAC
    select A[1:0], State select OP[1:0], Data bits D[11:8]
807                                                 SPI0_DL=data;  // Data bits D
    [7:0]
808
809                                             }
```

```
810
811
812
813                                                 if (nRepts < nRepetitions[3]){
814
815                                                   if(controller3[i] == SWITCHOFF
    && rep3!=0){
816                                                           IN13_Clr();
817                                                           IN23_Clr();
818                                                           rep3 = 0;
819                                                   }
820                                                   else if(controller3[i] ==
    SWITCHON01 && rep3!=1){ //positive phase
821                                                           IN13_Set();
822                                                           IN23_Clr();
823                                                           rep3 = 1;
824                                                   }
825                                                   else if(controller3[i] ==
    SWITCHON10 && rep3!=2){ //negative phase
826                                                           IN13_Clr();
827                                                           IN23_Set();
828                                                           rep3 = 2;
829                                                   }
830
831
832                                                         if (pwr > 0 && invRatio
    [3] == 1 && nRepts <= RepsMax[3]) {
833                                                                 current3[i] =
    current3[i] * (repetitonAmpRatio[3]/10);
834
835                                                         }
836                                                         else if (pwr > 0 &&
    invRatio[3] == -1 && nRepts <= RepsMax[3]) {
837                                                                 current3[i] =
    current3[i] / (repetitonAmpRatio[3]/10);
838
839                                                         }
840                                                         else { //included this so
     that the delay between repetitions wont be so big
841                                                                 current3[i] =
    current3[i] * ((repetitonAmpRatio[3]*10)/ (repetitonAmpRatio[3]*10));
842                                                         }
843
844                                                         // for the DAC C and when
     it is synchronous fire have to add address
845                                                 data = (current3[i] & 0xFF);
    // least significant byte (8-bit from the right)
```

```
846                                                 addressData = (current3[i] >>
    8) + 0x80;       //adds DAC:10 and OP:00
847
848                                                 while(!(SPI0_S));
849                                                 SPI0_DH=addressData;  //DAC
    select A[1:0], State select OP[1:0], Data bits D[11:8]
850                                                 SPI0_DL=data;  // Data bits D
    [7:0]
851
852                                               }
853
854
855
856                                               if (nRepts < nRepetitions[4]){
857
858                                                       if(controller4[i] ==
    SWITCHOFF && rep4!=0){
859                                                         IN14_Clr();
860                                                         IN24_Clr();
861                                                         rep4 = 0;
862                                                       }
863                                                       else if(controller4[i] ==
     SWITCHON01 && rep4!=1){ //positive phase
864                                                         IN14_Set();
865                                                         IN24_Clr();
866                                                         rep4 = 1;
867                                                       }
868                                                       else if(controller4[i] ==
     SWITCHON10 && rep4!=2){ //negative phase
869                                                         IN14_Clr();
870                                                         IN24_Set();
871                                                         rep4 = 2;
872                                                       }
873
874
875                                                       if (pwr > 0 && invRatio
    [4] == 1 && nRepts <= RepsMax[4]) {
876                                                               current4[i] =
    current4[i] * (repetitonAmpRatio[4]/10);
877
878                                                       }
879                                                       else if (pwr > 0 &&
    invRatio[4] == -1 && nRepts <= RepsMax[4]) {
880                                                               current4[i] =
    current4[i] / (repetitonAmpRatio[4]/10);
881
882                                                       }
```

```
883                                                 else { //included this so that the delay between repetitions wont be so big
884                                                         current4[i] = current4[i] * ((repetitonAmpRatio[4]*10)/ (repetitonAmpRatio[4]*10));
885                                                 }
886
887                                                 // for the DAC D and when it is synchronous fire have to add address and update outputs
888                                                 data = (current4[i] & 0xFF);    // least significant byte (8-bit from the right)
889                                                 addressData = (current4[i] >> 8) + 0xD0;        //adds DAC:11 and OP:01
890
891                                                 while(!(SPI0_S));
892                                                 SPI0_DH=addressData;  // DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
893                                                 SPI0_DL=data;    // Data bits D[7:0]
894
895                                         }
896
897
898                                         /// This condition is here for the case when stimulation in some channel is shorter than others, to reassure the output ///
899                                         /// the will be 0. These conditions are of most importance in the case where Channel 4 doesn't exist because otherwise it is ///
900                                         /// the one which updates DAC outputs and if stops occurring the digital order to update DAC outputs would also not occur ///
901
902                                         if (nRepts >= nRepetitions[1]){
903
904                                                 IN11_Clr();
905                                                 IN21_Clr();
906
907                                                 data = (0x00 & 0xFF); // least significant byte (8-bit from the right)
908                                                 addressData = (0x00 >> 8);    // shift to get the remaining 4-bits of the 12-bit
909                                                 // for the DAC A and when it is synchronous fire no need to add nothing to addressData
910
911                                                 while(!(SPI0_S));
912                                                 SPI0_DH=addressData; //DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
```

```
913                                                     SPI0_DL=data;  // Data
    bits D[7:0]
914
915                                             }
916
917                                             if (nRepts >= nRepetitions[2]){
918                                                     IN12_Clr();
919                                                     IN22_Clr();
920
921                                                     data = (0x00 & 0xFF);
    // least significant byte (8-bit from the right)
922                                                     addressData = (0x00 >>
    8) + 0x40;      //adds DAC:01 and OP:00
923
924                                                     while(!(SPI0_S));
925                                                     SPI0_DH=addressData;
    //DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
926                                                     SPI0_DL=data;  // Data
    bits D[7:0]
927                                             }
928
929                                             if (nRepts >= nRepetitions[3]){
930                                                     IN13_Clr();
931                                                     IN23_Clr();
932
933                                                     data = (0x00 & 0xFF);
    // least significant byte (8-bit from the right)
934                                                     addressData = (0x00 >>
    8) + 0x80;      //adds DAC:10 and OP:00
935
936                                                     while(!(SPI0_S));
937                                                     SPI0_DH=addressData;
    //DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
938                                                     SPI0_DL=data;  // Data
    bits D[7:0]
939                                             }
940
941                                             if (nRepts >= nRepetitions[4] ||
    nRepetitions[4] == 0){
942                                                     IN14_Clr();
943                                                     IN24_Clr();
944
945                                                     data = (0x00 & 0xFF);
    // least significant byte (8-bit from the right)
946                                                     addressData = (0x00 >> 8)
     + 0xD0;       //adds DAC:11 and OP:01
947
948                                                     while(!(SPI0_S));
```

```
949                                                     SPI0_DH=addressData;  //
    DAC select A[1:0], State select OP[1:0], Data bits D[11:8]
950                                                     SPI0_DL=data;    // Data
    bits D[7:0]
951                                                 }
952
953
954
955                                             }
956
957
958                                             //// WAIT FOR INTERSTIMULUS TIME //////
959                                             WAIT1_Waitus(interstimulusTimeSynch);
960                                             nRepts++;
961
962                                     }
963                             }
964
965                             // Full stimulation is applied on the chosen channels (
    repetition included with its own frequency) in sequence
966                         else if (firingType == 3){
967                                     nRepts = 0;
968
969                                     while (nRepts < nRepetitions[1]) {
970                                             i = 0;
971                                             rep = 0;
972                                             for (i = 0; i <= (n-1); i++) {
973
974                                                     if(controller1[i] == SWITCHOFF &&
     rep!=0){
975                                                             IN11_Clr();
976                                                             IN21_Clr();
977                                                             rep = 0;
978                                                     }
979                                                     else if(controller1[i] ==
    SWITCHON01 && rep!=1){ //positive phase
980                                                             IN11_Set();
981                                                             IN21_Clr();
982                                                             rep = 1;
983                                                     }
984                                                     else if(controller1[i] ==
    SWITCHON10 && rep!=2){ //negative phase
985                                                             IN11_Clr();
986                                                             IN21_Set();
987                                                             rep = 2;
988                                                     }
989
990
```

```
991                                             if (pwr > 0 && invRatio[1] == 1
    && nRepts <= RepsMax[1]) {
992                                                     current1[i] = current1[i]
     * (repetitonAmpRatio[1]/10);
993
994                                             }
995                                             else if (pwr > 0 && invRatio[1]
    == -1 && nRepts <= RepsMax[1]) {
996                                                     current1[i] = current1[i]
     / (repetitonAmpRatio[1]/10);
997
998                                             }
999                                             else { //included this so that
    the delay between repetitions wont be so big
1000                                                    current1[i] = current1[i]
     * ((repetitonAmpRatio[1]*10)/ (repetitonAmpRatio[1]*10));
1001                                            }
1002
1003                                            data = (current1[i] & 0xFF);
    // least significant byte (8-bit from the right)
1004                                            addressData = (current1[i] >> 8)
    + 0x10;        // shift to get the remaining 4-bits of the 12-bit
1005                                            // for the DAC A and when it is
    alone fire need to add DAC:00 and OP:01 to addressData
1006
1007                                            while(!(SPI0_S));
1008                                            SPI0_DH=addressData; //DAC select
     A[1:0], State select OP[1:0], Data bits D[11:8]
1009                                            SPI0_DL=data;   // Data bits D
    [7:0]
1010                                    }
1011
1012                                    IN11_Clr();
1013                            IN21_Clr();
1014
1015                                    data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
1016                                    addressData = (0x00 >> 8) + 0x10;
    // shift to get the remaining 4-bits of the 12-bit
1017                                    // for the DAC A and when it is alone
    fire need to add DAC:00 and OP:01 to addressData
1018
1019                                    while(!(SPI0_S));
1020                                    SPI0_DH=addressData;  //DAC select A
    [1:0], State select OP[1:0], Data bits D[11:8]
1021                                    SPI0_DL=data;    // Data bits D[7:0]
1022
1023                                    nRepts++;
```

```
1024                                    //// WAIT FOR INTERSTIMULUS TIME //////
1025                                    WAIT1_Waitus(interstimulusTime[1]);
1026                                }
1027
1028
1029                                //// WAIT FOR TIME_LAG ///
1030                                WAIT1_Waitms(timeLag);
1031
1032                                nRepts = 0;
1033                                while (nRepts < nRepetitions[2]){
1034
1035                                    i = 0;
1036                                    rep = 0;
1037                                    for (i = 0; i <= (n-1); i++) {
1038
1039                                        if(controller2[i] == SWITCHOFF &&
     rep!=0){
1040                                            IN12_Clr();
1041                                            IN22_Clr();
1042                                            rep = 0;
1043                                        }
1044                                        else if(controller2[i] ==
    SWITCHON01 && rep!=1){ //positive phase
1045                                            IN12_Set();
1046                                            IN22_Clr();
1047                                            rep = 1;
1048                                        }
1049                                        else if(controller2[i] ==
    SWITCHON10 && rep!=2){ //negative phase
1050                                            IN12_Clr();
1051                                            IN22_Set();
1052                                            rep = 2;
1053                                        }
1054
1055
1056                                        if (pwr > 0 && invRatio[2] == 1
    && nRepts <= RepsMax[2]) {
1057                                            current2[i] = current2[i]
     * (repetitonAmpRatio[2]/10);
1058
1059                                        }
1060                                        else if (pwr > 0 && invRatio[2]
    == -1 && nRepts <= RepsMax[2]) {
1061                                            current2[i] = current2[i]
     / (repetitonAmpRatio[2]/10);
1062
1063                                        }
```

```
1064                                            else { //included this so that
    the delay between repetitions wont be so big
1065                                                    current2[i] = current2[i]
     * ((repetitonAmpRatio[2]*10) / (repetitonAmpRatio[2]*10));
1066                                            }
1067
1068                                            data = (current2[i] & 0xFF);
    // least significant byte (8-bit from the right)
1069                                            addressData = (current2[i] >> 8)
    + 0x50;                 // shift to get the remaining 4-bits of the 12-bit
1070                                            // for the DAC B and when it is
    alone fire need to add DAC:01 and OP:01 to addressData
1071
1072                                            while(!(SPI0_S));
1073                                            SPI0_DH=addressData; //DAC select
     A[1:0], State select OP[1:0], Data bits D[11:8]
1074                                            SPI0_DL=data;    // Data bits D
    [7:0]
1075                                    }
1076
1077                            IN12_Clr();
1078                            IN22_Clr();
1079
1080                                    data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
1081                                    addressData = (0x00 >> 8) + 0x50;
    //adds DAC:01 and OP:00
1082
1083                                    while(!(SPI0_S));
1084                                    SPI0_DH=addressData;  //DAC select A
    [1:0], State select OP[1:0], Data bits D[11:8]
1085                                    SPI0_DL=data;    // Data bits D[7:0]
1086
1087                                    nRepts++;
1088                                    //// WAIT FOR INTERSTIMULUS TIME //////
1089                                    WAIT1_Waitus(interstimulusTime[2]);
1090                            }
1091
1092
1093                            //// WAIT FOR TIME_LAG ///
1094                            WAIT1_Waitms(timeLag);
1095
1096                            nRepts = 0;
1097                            while (nRepts < nRepetitions[3]){
1098                                    i = 0;
1099                                    rep = 0;
1100                                    for (i = 0; i <= (n-1); i++) {
1101
```

```
1102                                                    if(controller3[i] == SWITCHOFF &&
     rep!=0){
1103                                                            IN13_Clr();
1104                                                            IN23_Clr();
1105                                                            rep = 0;
1106                                                    }
1107                                                    else if(controller3[i] ==
    SWITCHON01 && rep!=1){ //positive phase
1108                                                            IN13_Set();
1109                                                            IN23_Clr();
1110                                                            rep = 1;
1111                                                    }
1112                                                    else if(controller3[i] ==
    SWITCHON10 && rep!=2){ //negative phase
1113                                                            IN13_Clr();
1114                                                            IN23_Set();
1115                                                            rep = 2;
1116                                                    }
1117
1118
1119                                                    if (pwr > 0 && invRatio[3] == 1
    && nRepts <= RepsMax[3]) {
1120                                                            current3[i] = current3[i]
     * (repetitonAmpRatio[3]/10);
1121
1122                                                    }
1123                                                    else if (pwr > 0 && invRatio[3]
    == -1 && nRepts <= RepsMax[3]) {
1124                                                            current3[i] = current3[i]
     / (repetitonAmpRatio[3]/10);
1125
1126                                                    }
1127                                                    else { //included this so that
    the delay between repetitions wont be so big
1128                                                            current3[i] = current3[i]
     * ((repetitonAmpRatio[3]*10)/ (repetitonAmpRatio[3]*10));
1129                                                    }
1130
1131                                                    data = (current3[i] & 0xFF);
    // least significant byte (8-bit from the right)
1132                                                    addressData = (current3[i] >> 8)
    + 0x90;                 // shift to get the remaining 4-bits of the 12-bit
1133                                                    // for the DAC C and when it is
    alone fire need to add DAC:10 and OP:01  to addressData
1134
1135                                                    while(!(SPI0_S));
1136                                                    SPI0_DH=addressData; //0xC3 //DAC
     select A[1:0], State select OP[1:0], Data bits D[11:8]
```

```
1137                                            SPI0_DL=data;   //0x3D // Data
    bits D[7:0]
1138                                    }
1139
1140                                IN13_Clr();
1141                                IN23_Clr();
1142
1143                                    data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
1144                                    addressData = (0x00 >> 8) + 0x90;
    //adds DAC:10 and OP:00
1145
1146                                    while(!(SPI0_S));
1147                                    SPI0_DH=addressData;  //DAC select A
    [1:0], State select OP[1:0], Data bits D[11:8]
1148                                    SPI0_DL=data;    // Data bits D[7:0]
1149
1150                                    nRepts++;
1151                                    //// WAIT FOR INTERSTIMULUS TIME //////
1152                                    WAIT1_Waitus(interstimulusTime[3]);
1153
1154                                }
1155
1156                                //// WAIT FOR TIME_LAG ///
1157                                WAIT1_Waitms(timeLag);
1158
1159
1160                                nRepts = 0;
1161                                while (nRepts < nRepetitions[4]){
1162
1163                                    i = 0;
1164                                    rep = 0;
1165                                    for (i = 0; i <= (n-1); i++) {
1166
1167                                        if(controller4[i] == SWITCHOFF &&
     rep!=0){
1168                                            IN12_Clr();
1169                                            IN22_Clr();
1170                                            rep = 0;
1171                                        }
1172                                        else if(controller4[i] ==
    SWITCHON01 && rep!=1){ //positive phase
1173                                            IN12_Set();
1174                                            IN22_Clr();
1175                                            rep = 1;
1176                                        }
1177                                        else if(controller4[i] ==
    SWITCHON10 && rep!=2){ //negative phase
```

```
1178                                                    IN12_Clr();
1179                                                    IN22_Set();
1180                                                    rep = 2;
1181                                                }
1182
1183
1184                                                if (pwr > 0 && invRatio[4] == 1
    && nRepts <= RepsMax[4]) {
1185                                                    current4[i] = current4[i]
     * (repetitonAmpRatio[4]/10);
1186
1187                                                }
1188                                                else if (pwr > 0 && invRatio[4]
    == -1 && nRepts <= RepsMax[4]) {
1189                                                    current4[i] = current4[i]
     / (repetitonAmpRatio[4]/10);
1190
1191                                                }
1192                                                else { //included this so that
    the delay between repetitions wont be so big
1193                                                    current4[i] = current4[i]
     * ((repetitonAmpRatio[4]*10)/ (repetitonAmpRatio[4]*10));
1194                                                }
1195
1196                                                data = (current4[i] & 0xFF);
    // least significant byte (8-bit from the right)
1197                                                addressData = (current4[i] >> 8)
    + 0xD0;                // shift to get the remaining 4-bits of the 12-bit
1198                                                // for the DAC D and when it is
    alone fire need to add DAC:11 and OP:01 to addressData
1199
1200                                                while(!(SPI0_S));
1201                                                SPI0_DH=addressData; //0xC3 //DAC
     select A[1:0], State select OP[1:0], Data bits D[11:8]
1202                                                SPI0_DL=data;   //0x3D // Data
    bits D[7:0]
1203                                        }
1204
1205                                IN14_Clr();
1206                                IN24_Clr();
1207
1208                                        data = (0x00 & 0xFF);   // least
    significant byte (8-bit from the right)
1209                                        addressData = (0x00 >> 8) + 0xD0;
    //adds DAC:11 and OP:01
1210
1211                                        while(!(SPI0_S));
```

```
1212                                        SPI0_DH=addressData;  //DAC select A
      [1:0], State select OP[1:0], Data bits D[11:8]
1213                                        SPI0_DL=data;    // Data bits D[7:0]
1214
1215                                        nRepts++;
1216                                        //// WAIT FOR INTERSTIMULUS TIME //////
1217                                        WAIT1_Waitus(interstimulusTime[4]);
1218                                    }
1219
1220                    }
1221
1222
1223                         // Turns OFF all the switches - just as an insurance
1224                    IN11_Clr();
1225                    IN21_Clr();
1226                    IN12_Clr();
1227                    IN22_Clr();
1228                    IN13_Clr();
1229                    IN23_Clr();
1230                    IN14_Clr();
1231                    IN24_Clr();
1232
1233                      // to power down the outputs
1234                      addressData = (current1[i] >> 8) + 0x70;    //adds DAC:00 and
       OP:11
1235                      while(!(SPI0_S));
1236                      SPI0_DH=addressData; //0xC3 //DAC select A[1:0], State select
       OP[1:0], Data bits D[11:8]
1237                      SPI0_DL=data;       //0x3D // Data bits D[7:0]
1238
1239                      //BLUE_SetVal(BLUE_DeviceData); /* turn off blue LED */
1240                      RED_SetVal(RED_DeviceData);
1241
1242                 }
1243
1244          }
1245
1246    /*** Don't write any code pass this line, or it will be deleted during code
         generation. ***/
1247    /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component.
         DON'T MODIFY THIS CODE!!! ***/
1248    #ifdef PEX_RTOS_START
1249      PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is
         defined by the RTOS component. */
1250    #endif
1251    /*** End of RTOS startup code.  ***/
1252    /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
1253    for(;;){}
```

```
1254   /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
1255 } /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
1256
1257 /* END main */
1258 /*!
1259 ** @}
1260 */
1261 /*
1262 ** ###################################################################
1263 **
1264 **     This file was created by Processor Expert 10.3 [05.09]
1265 **     for the Freescale Kinetis series of microcontrollers.
1266 **
1267 ** ###################################################################
1268 */
```