BE Project Report

Design of a closed-loop deep brain stimulation system for pre-clinical testing in animal models

Adil Dahlan

A thesis submitted in part fulfilment of the degree of

BE (Hons) in Biomedical Engineering

Supervisor: Prof. Madeleine Lowery

School of Electrical and Electronic Engineering
University College Dublin

May 20, 2020

BE (Biomedical) Degree Programmes

Stage 4 Project Report Receipt

Student Name: Adil Dahlan
Student Number: 16203187
Project Title: Designing closed loop deep brain stimulation system
Supervisor: Prof Madeleine Lowery

Plagiarism: the unacknowledged inclusion of another person's writings or ideas or formally presented work (including essays, examinations, projects, laboratory presentations). The penalties associated with plagiarism designed to impose sanctions seriousness of University's commitment to academic integrity. Ensure that you have read the University's ***Briefing for Students on Academic Integrity and Plagiarism*** and the UCD ***Statement, Plagiarism Policy and Procedures***, (http://www.ucd.ie/registrar/)

Declaration of Authorship
I declare that all of the following are true:
1) I fully understand the definition of plagiarism.
2) I have not plagiarised any part of this project and it is my original work.
3) All material in this report is my own work except where there is clear acknowledgement and appropriate reference to the work of others.

I acknowledge the contribution of the following post-graduate students/ post-doctoral fellows/ researchers or technicians to work detailed in this report:

Signed…Adil…… Date ……20/5/2020……

Office Use Only

Date and Time Received:	**Received by:**
[School Stamp]	
	Report Tracking
	Demonstrator:
	Lecturer:

Acknowledgments

I would like to deeply thank Professor Madeleine Lowery, my supervisor, for her continuous support and guidance, enthusiastic encouragement, constructive critiques and believing in me, giving me the chance to be part of this amazing project despite joining the course late.

Additionally, I would like to also thank MultiChannel Systems LLC team for lending us the W2100 system I used in this project and in particular I would like to thank Dr Jens, the Head of Firmware Department, for helping us reprogram the system to implement closed-loop stimulation.

I would like to also thank Mr John Fleming for helping me out with programming the system and Mr Jeremy Liegey for guiding me through the experimental testing protocol.

I would like to extend my thanks to Mr Declan Lehane and Mr Patrick McNally for helping me out in designing the experimental test equipment including circuits design and printing.

Last but not least, I would like to thank my Father, Imad, and my sister, Lian, as well as all my family members and fiends for their constant support throughout my college years.

Contents

Chapter 1: **Introduction**

Deep brain stimulation (DBS) has been established as a surgical therapy for treating Parkinson's disease since it was approved by the FDA in 2002 for its long-lasting effects. In clinical setup, DBS is used in a continuous, open-loop, configuration where the stimulation parameters, namely the stimulation frequency and pulse amplitude and width are kept always constant, operating 24/7, unless changed by the physician during a checkup appointment. However, due to the dynamic nature of the disease, patient's symptoms change throughout the day depending on what time it is, the mood of the patient and also his physical state. Thus, there has been huge interest over the past few years in developing closed-loop control algorithms that would change the stimulation parameters accordingly, also known as adaptive DBS (aDBS).

The Neuromuscular Systems Research Group, under the supervision of Professor Madeleine Lowery, have proposed the "Proportional-Integral Control" algorithm for aDBS which uses the beta-band activity recorded from the subthalmic nucleus (STN) in the brain as a biomarker for modulating the stimulation parameters. This algorithm has been shown computationally to provide more effective suppression of pathological activity with lower power consumption compared to other existing aDBS control algorithms. As part of the path towards translating this control algorithm to patient, it is necessary to test it first in pre-clinical animal models.

Since the "Proportional-Integral Control" algorithm tolerates a delay of around $100ms$ between recording the beta-band activity and modulating the stimulation parameters accordingly. The aim of this project is to design and implement a closed-loop deep brain stimulation system for pre-clinical testing in animal models of Parkinson's disease, along with quantifying all system's delays. The system should be able to simultaneously record the neural activity from the brain, namely the beta-band LFP activity from the STN, and change, in real-time, the stimulation pulse as per determined by the closed-loop control algorithm . For ethical considerations, ideally, the system should have a wireless connection between the headstage (the unit implanted on the head of the animal model) and the signal processing unit.

Chapter 2: **Literature Review**

Parkinson's disease (PD) is a neurodegenerative disorder caused by the death of dopaminergic cell death in the substantia nigra pars compacta (SNpc), which leads to the decreased striatal inhibition to the cortico-basal-ganglia network that is hypothesized to cause the motor symptoms associated with PD [2].

There are four cardinal motor symptoms associated with PD: tremor, rigidity, bradykinesia, and postural instability (or freezing of gait). Typically, Parkinsonian tremor is a type of rest tremor that occurs when the affected limb is at rest (at frequencies from 2-7 Hz [47]). PD patients may also exhibit postural tremor arising when one is trying to maintain limb posture in a certain position (as when reaching out to shake someones hand) and also kinetic tremor being present when performing a visually-guided movement (such as eating or writing) to varying degrees. Rigidity and bradykinesia describe slowness of movement and difficulty performing repetitive movements. Postural instability and freezing of gait is often more difficult to characterize as symptomatic episodes are generally more intermittent and context-dependent (walking indoors or outdoors).Neurologists assess PD symptom severity using the United Parkinson's Disease Rating Scale (UPDRS) [24]

How basal ganglia dysfunction leads to parkinsonism is not yet entirely clear. Depth recordings in patients with Parkinson's disease (PD) have established the existence of synchronization within neuronal populations of the basal ganglia in several different frequency bands [16] [35] [50] [32], leading to a new schema for basal ganglia disorders in which the nature of synchronized bursting within the basal ganglia is critical in determining parkinsonism [41] [13]. This excessive synchronization tends to dominate in the beta band (14–35 Hz) LFP recorded from the subthalamic nucleus (STN) of the basal ganglia in patients with PD [13]. It has been proven that the level of activity in the beta band LFP recorded from the STN level correlates with PD symptoms such as motor impairment (with and without treatment), bradykinesia [38] and reaction times [34].

As PD has no cure, all treatments currently available are palliative and serve merely to improve

quality of life of the patients. Typically, PD patients are treated first with the pharmaceutical Levadopa, which effectively increases dopamine concentrations in the brain in order to offset the dopamine-deficient SNpc [36]. However, for many PD patients, treatment with Levadopa alone is often unable to manage the worsening symptoms as many patients eventually become tolerant to these drugs, at which point neurosurgery for implantation of a DBS is recommended, which has been granted FDA approval in 2002 for its long-lasting beneficial effects. [18]

A DBS system consists of an infraclavicular implanted pulse generator (IPG) which is connected to electrodes placed in deep brain structures. The most common anatomical targets of DBS for treatment of PD are the internal globus pallidus (GPi) and the STN. After implantation, the physician fine-tunes the pulse amplitude value, in the range between $0mA$ and $3mA$, in order to maximize the alleviation of PD symptoms while minimizing the unintended side effects of stimulation. The pulse width and frequency of stimulation are typically maintained invariable at around $60\mu s$ and $130Hz$ respectively. This manual parameter optimization process can take several months [55] and stimulation parameters typically need to be changed by the clinician every 3-12 months [55].

Despite much research, the mechanism by which DBS exerts its therapeutic effects is not well understood. It has been observed that high-frequency DBS of the STN attenuates the neural activity in the alpha/beta band resulting in decreased PD symptoms [64, 50, 12, 14, 65, 66, 21, 23, 30, 53, 31, 39, 33, 52, 42], where a decreased beta power occurs before and during movement [35] [49] [34] [20]. It is generally thought that DBS overrides the abnormal neuronal activity in these areas and facilitates non-pathological oscillatory activity [10, 45, 44].

In many brain disorders such as PD, symptoms fluctuate on a moment-by-moment basis depending on factors such as cognitive and motor load and concurrent drug therapy. Existing DBS systems do not take into account the variable nature of symptoms that result from neurological movement disorders [27, 60]. Clinical DBS therapy is limited to 'open-loop' neurostimulation: the neurostimulator does not sense the brain signals nor the behavior it is modulating. It applies continuous pulse trains of fixed frequency, amplitude, pulse width, and pattern, and cannot adjust such parameters in response to neural activity, or the patient's state of activity or behavior. With

few exceptions, DBS is applied at regular and high frequencies, in excess of $100Hz$. Such high frequencies of stimulation are believed to create an acute information lesion, whereby transmission of aberrant neural activity is attenuated [25, 3, 28]. At the same time, any residual physiological communication may be impaired in the stimulated brain circuit [19] resulting in several side effects such as speech impairment, tingling, numbness, involuntary muscle contractions, psychiatric symptoms and antagonistic worsening in some motor functions. [19, 8, 62, 68, 29, 17, 5]

Emerging evidence suggests that non-continuous adaptive DBS (aDBS), or so called closed-loop DBS (clDBS), may be more efficacious and efficient with fewer adverse effects than open loop DBS (olDBS). [37, 39, 54] In aDBS, stimulation parameters are adjusted in 'real time' based on the severity symptoms and the current state of the PD patient including physical activity, body position, the sleep/wake cycle, and overall progression of the disease [39, 11, 1, 54, 51, 9]. The primary goal of aDBS is to increase the specificity of the intervention, thereby widen the therapeutic window of neurostimulation improving the quality of life of the patient by maintaining an optimal stimulation. Secondary goals include reducing the costs associated with frequent visits to the treating physician and neurologist as well as reducing power drains on the IPG. Improvements in battery technology have lessened but not negated the gains to be had from reduced power consumption. A significant proportion of patients are unsuitable for rechargeable IPG systems[59], and those that are would benefit from the ability to recharge less frequently. However, perhaps the most exciting impact of reduced power demands might be the possibility of reducing rechargeable battery size sufficiently to enable skull mounted IPGs [26, 43, 56].

Successful STN aDBS in PD requires the discovery of patient specific 'biomarkers' which are neural and kinematic features that reflect the disease, state of activity and/or dominant symptom of the patient, along with a control algorithm that will change the stimulation parameters in a manner that modulates the stimulation parameters accordingly in order to alleviate PD symptoms. The ideal aDBS system will be able to concurrently sense and stimulate using a fully embedded sensing neurostimulator, which will automatically adapt the stimulation parameters (pulse amplitude and width, and stimulation frequency) based on a freely moving patient's state of activity and dominant symptom [63]. Finding the right 'biomarkers' and determining the best control algorithm for optimised PD aDBS therapy has been the focus of several research groups in the past years

[27, 38, 51].

Several biomarkers can be used to infer the current state of activity in the pathological circuits of the brain for the purpose of modulating the aDBS parameters; information can be derived from peripheral measures of motor state, such as the monitoring of tremor or involuntary writhing movements (dyskinesias), directly from recordings of brain activity, or through a combination of these approaches. [31, 57, 61, 33, 46]

LFP recordings from STN-DBS electrodes have previously revealed exaggerated beta band synchronization in PD patients that is reduced by levodopa and STN-DBS improving bradykinesia and rigidity. This has led to the suggestion that STN beta band oscillations are very good 'biomarkers' of the Parkinsonian state [31, 30, 53, 38, 12, 58] and thus have potential to be used as a biomarker for aDBS. Indeed, beta-band power in STN LFPs has already been used to trigger DBS in PD patients in an acute closed-loop system with promising results improving motor outcome and reducing energy consumption [39, 37, 40] and DBS side-effects such as dyskinesia [54] when compared with continuous DBS.

aDBS control algorithms can be categorized into either *amplitude modulation control algorithm* or *frequency modulation control algorithm*. Amplitude modulation control relies on changing the amplitude of pulses while maintaining pulses width and frequency of stimulation constant over time, usually at around $60\mu s$ and $130Hz$, respectively (similar to open-loop DBS). Whereas frequency modulation control relies on modulating the stimulation frequency while maintaining pulse length and amplitude constant over time, usually at around $60\mu s$ and $1.5mA$ respectively (similar to open-loop DBS). [22]

Various research groups developed different amplitude modulation algorithms, which have been tested on PD patients showing promising results. The study done by Simon Little *et al.* at University of Oxford utilized a relatively simple paradigm in which beta-band power crossing a heuristically determined threshold triggered stimulation, implementing what so called an "On-Off Control". [39, 37] Building up on that, the study done by Velisar *et al.* at Stanford University developed the "Dual-Threshold Control" algorithm where a lower and upper thresholds on the beta band power were experimentally determined such that beta-band oscillations between these

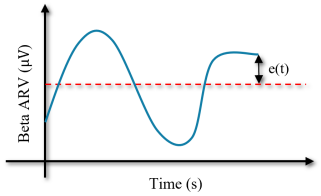
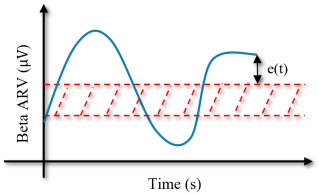
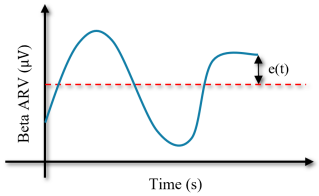
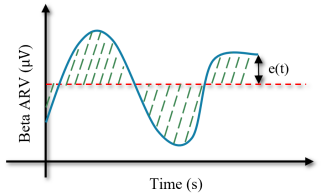
two threshold would be maintained through maintaining the same stimulation amplitude, whereas oscillations above the upper threshold and below the lower threshold triggered increasing and decreasing the stimulation amplitude, respectively. [63] This control algorithm was developed in order to suppress long pathological beta oscillations, greater than 400 ms which are closely related to motor symptoms of PD while also enhancing short beta-band burst which have been proven to be neurologically healthy. [4] Whereas the study done by Alberto Priori *et al.* at Università di Milano utilized a "Proportional Control" algorithm where the stimulation amplitude is proportional to the difference between a predetermined threshold and the average rectified signal beta-band signal [51]. Other studies have utilized more complex machine learning and feature extraction methods to detect the occurrence of tremor from beta-band power, as well as power in other frequency bands [7, 6, 15, 48, 67]. However, in all of these studies, signals were recorded intra- and peri-operatively.

The Neuromuscular Systems Research Group under the supervision of Professor Madeleine Lowery at University College Dublin has proposed a new algorithm that can be used for both amplitude and frequency modulation control, namely the "Proportional-Integral Control" algorithm. Using computational modelling, this algorithm has shown significant improvement in reducing the pathological beta-band activity in the STN, along with reducing the mean power consumption and error obtained compared to existing aDBS control algorithms. [22]

Table 2.1 outlines a brief summary of the various aDBS control algorithms discussed above.

In order to assess experimentally the performance of "Proportional-Integral Control" algorithm, the Neuromuscular Systems Research Group are proposing to test this algorithm on animal models, namely rats. Since the "Proportional-Integral Control" algorithm tolerates around 100ms of delay between recording the beta-band activity and modulating the stimulation parameters accordingly. The aim of this project is to design and implement a closed-loop deep brain stimulation system for pre-clinical testing in animal models of Parkinson's disease, along with quantifying all system's delays.

Table 2.1: Control Algorithms [22]

	On-Off Control	Dual-Threshold Control	Proportional Control	Proportional-Integral Control
Beta Average Rectified Value (ARV)				
Modulated DBS Parameter Value (u(t))	$u(t) = \begin{cases} u(t-1) + u_{\Delta R_L}; & e(t) > 0 \\ u(t-1) - u_{\Delta R_L}; & e(t) < 0 \end{cases}$ (2.1) where $u_{min} \leq u(t) \leq u_{max}$	$u(t) = \begin{cases} u(t-1) + u_{\Delta R_L}; & e(t) > 0 \\ u((t-1); & e(t) = 0 \\ u(t-1) - u_{\Delta R_L}; & e(t) < 0 \end{cases}$ (2.2) where $u_{min} \leq u(t) \leq u_{max}$	$u(t) = K_p e(t)$ (2.3) where $u_{min} \leq u(t) \leq u_{max}$	$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int e(\tau) d\tau \right)$ (2.4) where $u_{min} \leq u(t) \leq u_{max}$

$u(t)$: stimulation amplitude for amplitude modulation algorithms (on-off control, dual-threshold control, proportional control, proportional-integral control)

stimulation frequency for frequency modulation algorithms (proportional control and proportional-integral control)

$e(t)$: controller error input signal at time t; difference between Beta ARV and threshold value

$u_{\Delta R_L}$: rate limit of the DBS parameter at each controller call

K_p: proportional gain

T_i: controller integral time constant

u_{min}: minimum bound of the modulated DBS parameter

u_{max}: maximum bound of the modulated DBS parameter

Chapter 3: **Methods**

3.1 System Used

3.1.1 System Overview

The only commercially available system for wireless stimulation in rodents is the **W2100 Wireless System** by **MultiChannel Systems** (`https://www.multichannelsystems.com/products/wireless-systems`). It also also allows for simultaneous recording on up to 32 channels. Thus, this system could potentially be used for closed loop stimulation where the beta-band activity would be recorded and the stimulation pulse would be changed according. However, the system currently only allows for continuous (open-loop stimulation) and threshold-triggered stimulation with only one single pulse. Thus, the objective for this project was to reprogram the system to enable the stimulation pulse to be adjusted in real-time as determined by the closed-loop control algorithm. I also needed to quantify the delays associated with the system, namely the *recording*, *signal processing* and *stimulation delays* as the proportional-integral control algorithm tolerates delays of up to around $100ms$, as discussed previously.

3.1.2 System Components

The W2100 Wireless System comes with the following components:

1. **Headstage:** shown in figure 3.1, is the unit to be implanted onto the head of the animal model (rat).

(a) Bottom-view (b) Top-view (c) Side-view

Figure 3.1: W2100 wireless system headstage

The headstage is composed of:

- 4, 8, 16 or 32 recording electrodes (depending on the model used) of a sampling rate equal to $20KHz$.
- 2 stimulation electrodes delivering $-0.5mA$ to $0.5mA$ stimulation current.
- 101 gain amplifier; 16 bit resolution ADC; receiving and transmitting antennae to send recorded signals and receive stimulation pulses to and from the receiver respectively.

2. **Receiver:** shown in figure 3.2, connected to the headstage through RF link and to the interface board through an eSATAp port.

Figure 3.2: W2100 wireless system receiver

3. **Interface Board (IFB):** shown in figure 3.3, is connected to the receiver and the PC. The IFB contains mainly a digital signal processor (DSP) that processes the recorded signal and determines the pulse pattern to be applied at the stimulating electrodes of the headstage using a predetermined feedback control algorithm.

   The IFB accepts has 4 digital external inputs and 8 analog inputs sampled using a 32 Bit ADC. These external inputs can be used to record a TTL trigger signals, recorded video, temperature and voltage to facilitate the synchronization of all experimental data.

Figure 3.3: W2100 wireless system IFB

4. **Guided User Interface (GUI):** MultiChannel (MC) Systems provide a software package called *MultiChannel Suite*, which includes several softwares, whose icons are shown in 3.4:

   (a) **MC Experimenter** for combining virtual instruments (e.g. recorder, filter, spike detector, and much more) by drag and drop; designing the stimulation patterns for all system-integrated stimulus generators and viewing recorded signals online.

   (b) **MC Analyzer** facilitates offline analysis of data recorded using MC Experimenter.

   (c) **MC DataManager** which exports recorded data along with the analog and digital inputs into HDF5 format (for analysis in Matlab, Python, and other applications); .nex format for analysis in Neuroexplorer; and .ced Spike2 format.

(a) (b) (c)

Figure 3.4: (a) MC Experimenter icon; (b) MC Analyzer icon; (c) MC DataManager icon

3.1.3 Stimulation Control Options

Apart from continuous (open-loop) stimulation that can be set using the MC Experimenter GUI, the W2100 wireless system offers 2 possible control options to implement closed-loop stimulation which are outlined in table 3.1.

Table 3.1: W2100 Wireless System Closed-Loop Stimulation Options

	Threshold Triggered	**DSP Triggered**
Pulse Patterns	1	16
Stimulation Trigger	Threshold	1- DSP processes recorded signal
		2- DSP determines 1 of the 16 stimulation pattern to apply
		3- DSP "calls" 1 of 16 stimulation patterns stored in the FPGA on the headstage to be applied
Setup	MC Experimenter GUI	DSP programming
Expected Delays	Few ms	Tens of ms

3.1.4 System Delays

There are several delays associated with the W2100 wireless system, shown in figure 3.5, that I needed to quantify, namely being:

1. **Recording Delay:** This delay, defined as the delay the delay between recording the signal and receiving it in the DSP, is due to two reasons. First, the sampled recorded signal is sent in packages of 16 frames from the headstage to the receiver. Second, there is a transmission delay associated with the RF connection between the headstage and the receiver.

2. **DSP Delay:** the delay associated with processing the recorded signal and determining whether to stimulate or not and which pulse to apply. For both stimulation options; threshold triggered and DSP triggered, this delay is very small, less than the sampling period as informed by MultiChannel Firmware support team.

3. **Stimulation Delay:** This delay is expected to vary depending on whether we are constantly stimulating with the same pulse pattern (in the case of threshold triggered stimulation) or switching between different stimulation patterns (in the case of DSP triggered stimulation)

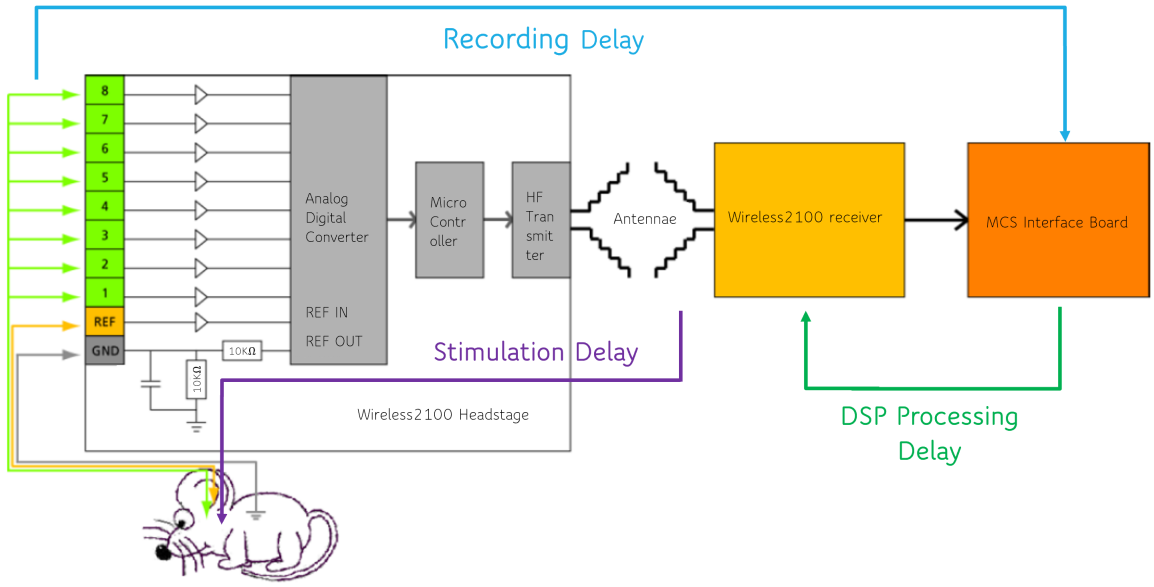


Figure 3.5: System Setup and Associated Delays

3.2 Threshold-Triggered Stimulation

3.2.1 Hardware Setup

As per recommended by MultiChannel Suite Firmware Support Team, I started off by implementing the threshold-triggered stimulation option and quantifying the *recording*, *DPS* and *stimulation* delays associated with it. To do so, I decided to carry out the following experiment, shown in figure 3.6:

1. Connect an external rectangular wave generator of amplitude $4.3mA$ to recording electrode 6 of the headstage.
2. Record the voltage across the external signal generator as an analog input to the IFB (port1).
3. Connect stimulating electrode 1 of the headstage through a $50km\Omega$ resistance in series with a 10Ω resistance to the ground electrode of the headstage.
4. Record the voltage across the 10Ω resistance as an analog input to the IFB (port2).

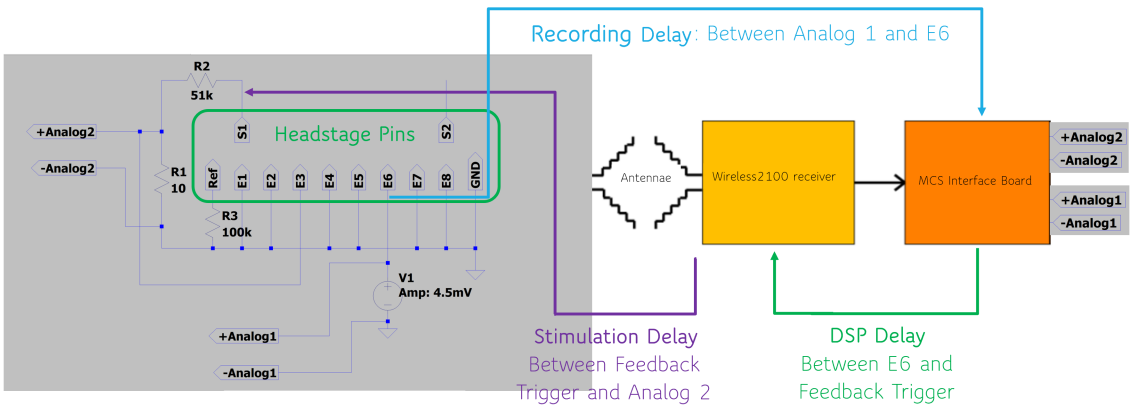


Figure 3.6: Experimental Setup and Associated Delays

NB: The $50k\Omega$ resistance between the stimulating and ground electrodes was included in series with the 10Ω resistance in order to mimic the setup in which the system will be operating in service (the impedance of the STN is approximately $50k\Omega$).

Experimental Equipment

To carry out the experiment outlined in 3.2, I used the following equipment:

1. **Headstage Connectors**, shown in figure 3.7, which I ordered from *Genalog Ltd* in order to connect the headstage to the interface board;

    - Through Hole: A79038-001 (NPD-18-DD-GS) connector,
    - Vertical Surface Mount: A79042-001 (NPD-18-VV-GS) connector,
    - Cable: A79044-001 (NPD-18-WD-18.0-C-GS) connector,

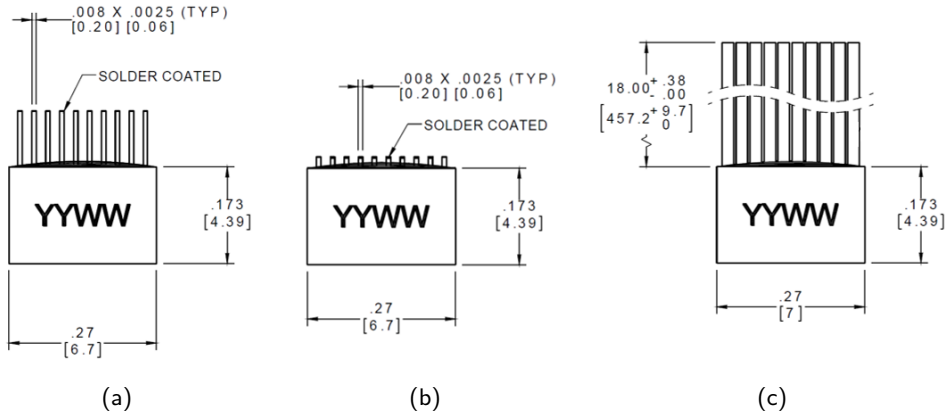


Figure 3.7: Headstage Connectors: (a) Through Hole Connector, (b) Vertical Surface Mount Connector, (c) Cable Connector

2. **2 Lemo Connectors**, shown in figure 3.8, which I ordered from *MultiChannel Systems* in order to connect the analog inputs to the IFB.

Figure 3.8: Lemo Connector for Analog Input to the IFB

3. **External Signal Source**, shown in figure 3.9, along with a load of 4.7Ω connected in parallel with its output in order to provide a $4.3mV$ rectangular wave recorded by the headstage. (the output voltage of the signal generator is $50mV$ and its internal output impedance is 50Ω. Thus, the output voltage recorded when the load of 4.7Ω is connected in parallel with its output is $\frac{50mV \times 4.7\Omega}{50\Omega + 4.7\Omega} = 4.3mV$)

(a) (b)

Figure 3.9: (a) Signal Generator and (b) Load Resistance

4. **Expansion Board**, shown in figure 3.10, onto which the headstage connector was soldered, used in order to provide easy access to the headstage electrodes.

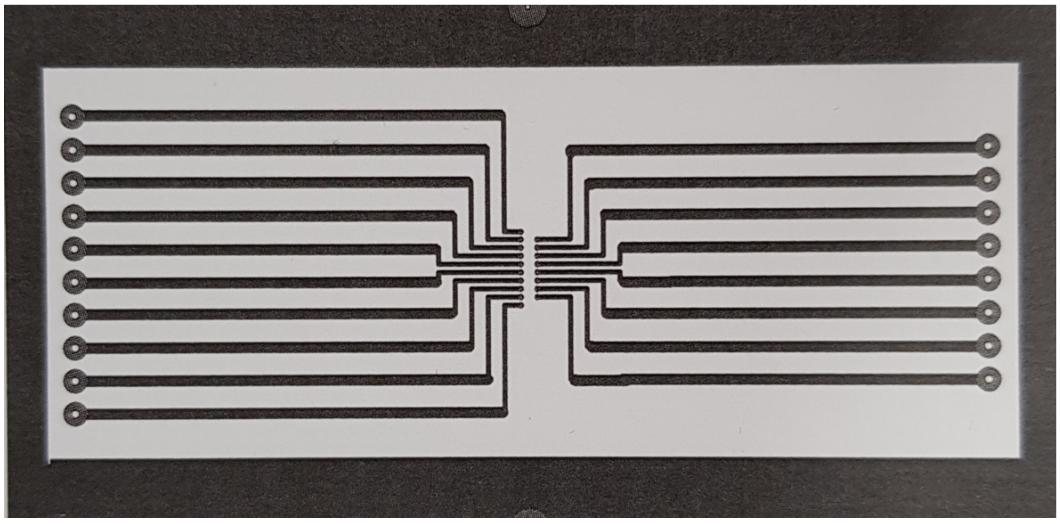


Figure 3.10: Headstage Expansion Board Diagram

3.2.2 Software Setup

Using MultiChannel Experimenter GUI, I set the stimulation pulse to be $2ms$ long and $0.5mA$ in amplitude, as shown in figure 3.11.

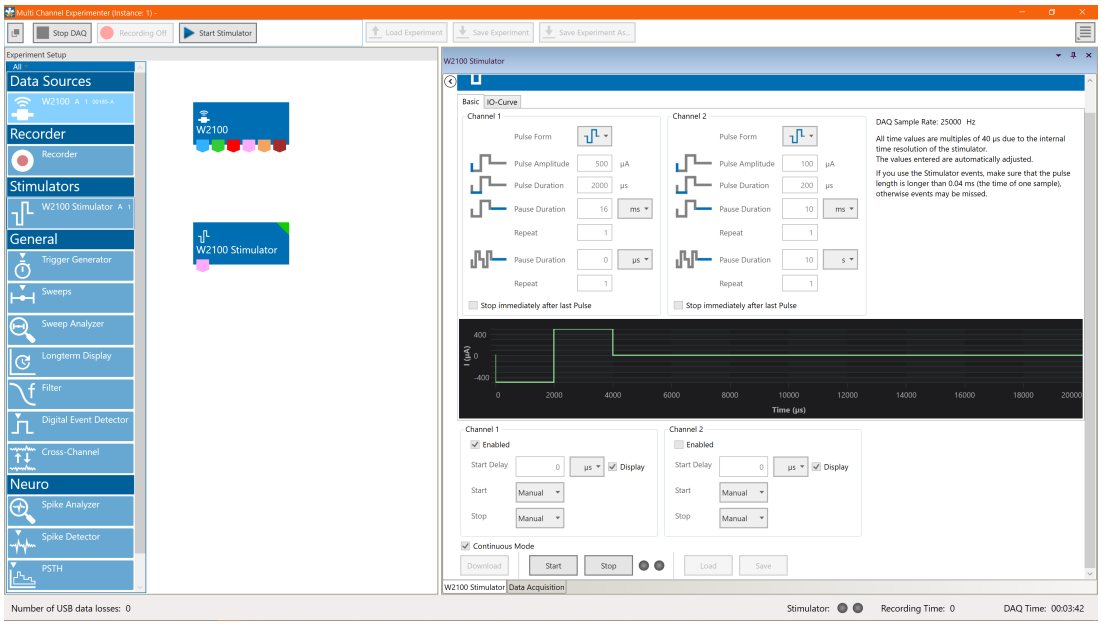


Figure 3.11: Stimulation Pulse Setup Using MC Experimenter GUI

I also set the feedback stimulation to be triggered when the recorded signal exceeds $1.448mV$, as shown in figure 3.12.

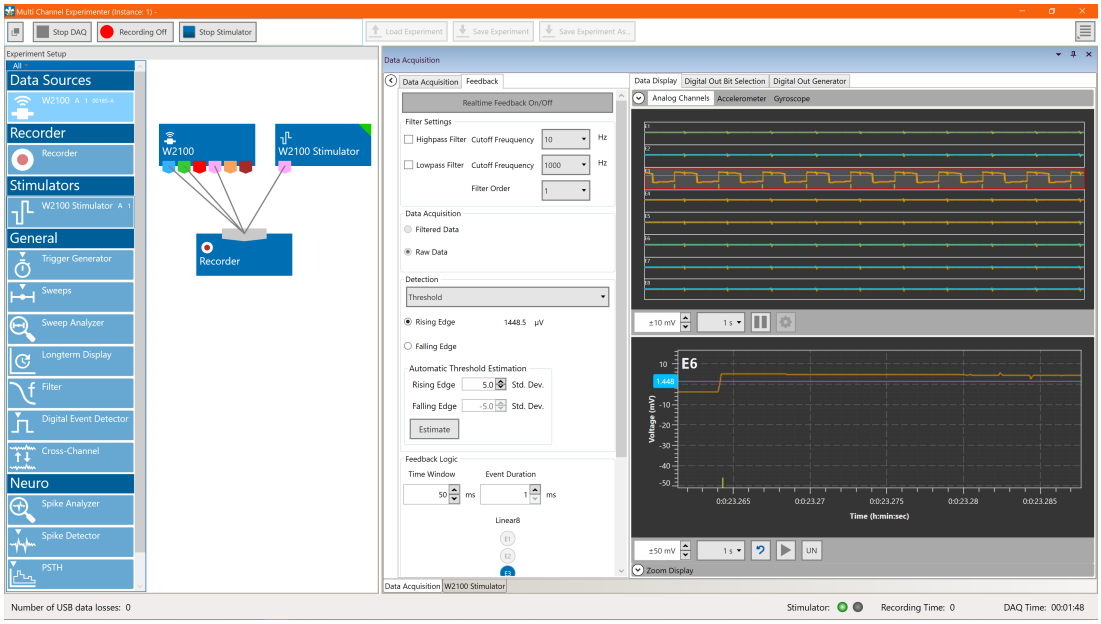


Figure 3.12: Threshold-Triggered Stimulation Setup Using MC Experimenter GUI

3.2.3 Delays Quantification

Recording Delay

To quantify the recording delay, I compared the delay between the recorded signals from electrode number 6 and analog input port1 of the IFB (which are both connected to the the external signal generator).

DSP Delay

To quantify the processing delay, I used the data obtained from the "trigger" data port at the simulator unit in the MC Experimenter GUI (the pink port, shown in figure 3.13) which marks when the start and stop commands are given to the simulator after the DSP had processed the recorded signal.

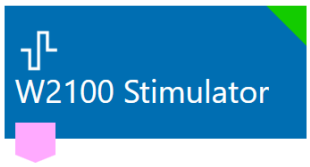


Figure 3.13: "Trigger" Port on the Stimulator Unit in MultiChannel Experimenter GUI

Knowing when the feedback should be triggered (i.e. when the recorded signal went above the threshold that should trigger feedback stimulation) and comparing that with when the stimulator was given the command to turn on (the data from the "trigger" port) I quantified the DSP delay.

Transmission Delay

To quantify the transmission delay I compared the data from the "trigger" port (i.e. when the stimulator was told to turn on) with when I saw an output at the stimulating electrode.

3.3 DSP-Triggered Stimulation

In order to implement closed-loop stimulation, Dr Jens, the Head of Firmware Development at MultiChannel Systems LLC, reprogrammed the W2100 system's firmware such that 16 pulse patterns could be stored onto the FPGA of the headstage and using any algorithm, one could switch between them. He also provided us with the system's code, in the (`McsDspRealtimeFeedback`) folder (`https://github.com/multichannelsystems/McsDspRealtimeFeedback`), which I used to implement DSP-triggered stimulation.

Below, I will first discuss how to program the system (3.3.1), namely:

- How to define the 16 pulse patterns.
- How to code the control algorithm.

Then, I will discuss the 4 different control algorithms I coded into the system (3.3.2), namely the *on-off* , *dual-threshold*, *proportional-integral (PI) amplitude and frequency modulation* control algorithms.

Finally, I will discuss how to set up the W2100 system for DSP-triggered stimulation (3.3.3), namely:

- How to configure the system.
- How to record data.
- How to compile the DSP code in order to obtain the binary file associated with it.
- How to upload the pulse patterns and start data acquisition.
- How to upload the binary code file into the DSP.

3.3.1 W2100 Programming for DSP-Triggered Stimulation

Pulses Declaration

The 16 different pulse pulses are declared in the `McsDspRealtimeFeedbac/Windows/DSPDacqExample/Form1.cs` file, namely inside the `startDacq_Click` function.

Each pulse is composed of 3 segments where each segment (x) is defined by its amplitude (*amp[x]* in μA) and duration (*dur[x]* in μs) as shown in figure 3.14

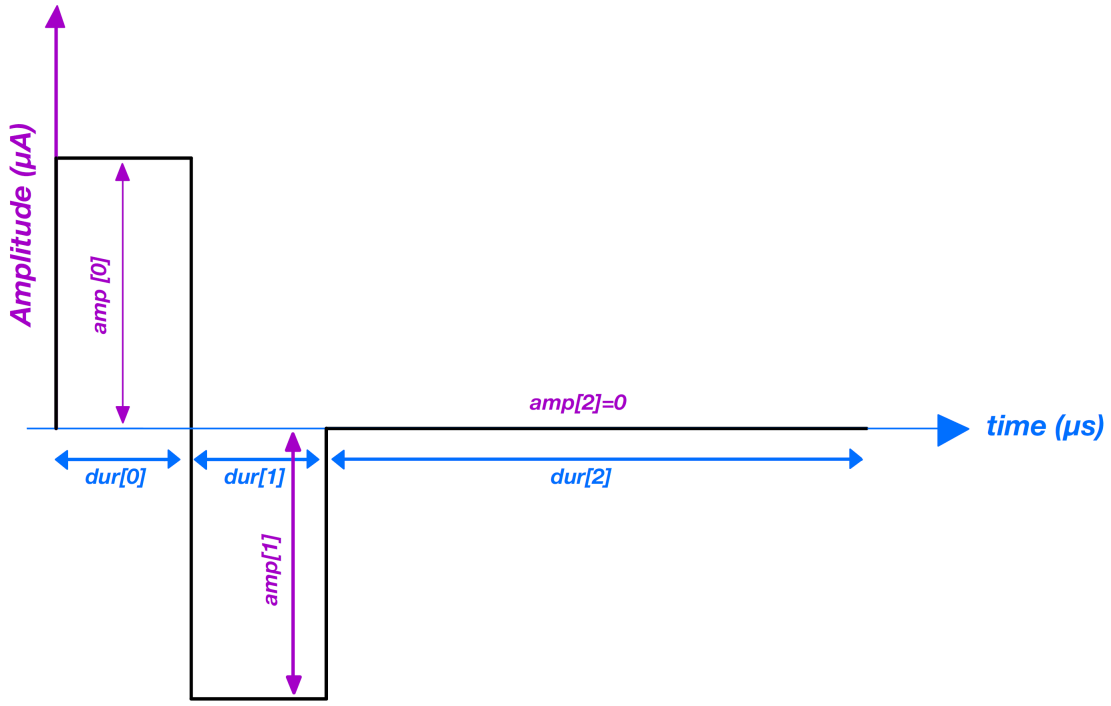


Figure 3.14: Pulse Parameters

Below is the section of the `startDacq_Click` function associated with defining the 16 pulses. After declaring the amplitude and duration of each of the 3 segments associated with each pulse, the pulse is created then written into the memory of the headstage (after checking that it doesn't exceed the memory space allocated to it).

```
143 private void startDacq_Click(object sender, EventArgs e)
144 {
        ...
228     // Define the amplitude vector of the 3 segments of the biphasic pulse (in uA)
229     int[] ampl = new[] { 1000000, -1000000, 0 };
230
231     // Define the duraion vector of the 3 segments of the biphasic pulse (in us)
232     ulong[] dur = new ulong[] { 2000, 2000, 10000 - 2 * 2000 };
233
```

```
234         // Define each pulse
235         for (int i = 0; i < 16; i++)
236         {
237             // Define the amplitude (uA) of each of the 3 segments (below is a random example)
238             ampl[0] = 40000 * i + 50000;
239             ampl[1] = -40000 * i - 50000;
240             ampl[2] = 0;
241 
242             // Define the duration (us) of each of the 3 segments (below is a random example)
243             dur[0] = (ulong)(2000 * i);
244             dur[1] = (ulong)(2000 * i);
245             dur[2] = (ulong)((dur[0] + dur[1]) * 2);
246 
247             // Define the associated pulse
248             CStimulusFunctionNet.StimulusDeviceDataAndUnrolledData prep = stim.PrepareData(0,
                ampl, dur, STG_DestinationEnumNet.channeldata_current, 1);
249 
250             // Check the available memory in the headstage
251             if (first)
252             {
253                 first = false;
254                 preplegth = prep.DeviceDataLength;
255             }
256 
257             // Check that the pulse fits into the designated memory
258             Debug.Assert(preplegth == prep.DeviceDataLength);
259             Debug.Assert(prep.DeviceDataLength <= 15);
260 
261             // Store pulse into designated memory
262             stim.SendPreparedData(0x10 * i + 0, prep,
                STG_DestinationEnumNet.channeldata_current);
263         }
            ...
278 }
```

Closed-Loop Control Algorithm Coding

When data acquisition is turned on, the sampled data is sent in packets of 16 frames, from the headstage, to the W2100 wireless system receive. However, once a packet of data is received, each data-frame is passed separately into the DSP at the same rate in which the data was sampled $(20000Hz)$. Each time a new data frame is passed into the DSP, the `interrupt6` function in `McsDspRealtimeFeedback\DSP\FB_W2100_SCU_MEA256\irq.c` file is called.

The closed-loop control algorithm was coded into the `interrupt6` function where the recorded signal is processed and based on that the stimulation pulse is chosen. Since the closed-loop algorithm is coded into one function running on the DSB then there's no stimulation delay associated with it, as informed by Dr Jens. However, I still need to quantify the stimulation delay.

I will discuss below how to manipulate `interrupt6` function:

- **Headstage Recorded Signal:** The *HS1_Data_p[x]* variable stores the sampled data from recording electrode *x* (in pV). Since the headstage we have has 8 recording electrodes then *x* ranges between 0 and 7.

- **Analog Input Data:** The *IF_Data_p[x]* variable stores the sampled analog input to the W2100 system interface board (in pV). Since the interface board has 8 analog inputs then *x* ranges between 0 and 7.

- **AUX Port Output:** The W2100 interface board has two AUX digital output ports which can be set high or low using the following line of code *WRITE_REGISTER(IFB_AUX_OUT, aux_value);* where *aux_value* equals 1 or 0. By setting the AUX port high before changing the stimulation pulse pattern and quantifying the delay between when the AUX port goes high and when the stimulation pulse changes we can quantify the stimulation delay.

- **Changing Stimulation Pulse:** *WRITE_REGISTER(0x9A80, 0x1000 * i + 0x100);* command-line orders the headstage to stimulate using the pulse whose index is *i*. Where each of the 16 pulses has it own index *i*, between 0 and 15, as previously defined in the `startDacq_Click` function. At the end of the pulse *i*, the headstage checks if the stimulation pulse needs to be changed (i.e. it checks if the same command line has been used with another value for *i*), if so, then it changes the stimulation pulse accordingly. Otherwise, it stimulates again with the same pulse it just finished. This goes on continuously until the headstage is commanded to stop stimulating, using the following command-line *WRITE_REGISTER(0x9A80, 0);*.

- **MonitorData:** As I will discuss later, when running the W2100 system using DSP-triggered stimulation, there's a simple GUI that can be used to plot data which include the analog and digital inputs to the IFB, the headstage recorded data and also any other 64 variables (*MonitorData[0]-MonitorData[63]*) that were defined in the `interrupt6` function. These MonitorData variables could be useful for plotting variables that were declared inside the DSP and were used in implementing the closed-loop control algorithm, such as the integral term, the error term (the difference between the recorded and target value beta average rectified value), the applied pulse amplitude / frequency or the one that would have been

applied if we were able to use the algorithm in the continuous form.

Below is a section of the `interrupt6` function where I ask the headstage to switch stimulation pulse every second, going through the different 16 pulses stored in the headstage, while setting the AUX port to high then low, before and after sending the command to switch stimulation pulse respectively.

```
111 interrupt void interrupt6(void)
112 {
113     // Define counter for the function call
114     static int timestamp = 0;
115 
116     // Define a variable to store the index of the pulse to apply
117     static int seg = 0;
        ...
311     // Increment timestamp eachtime interrupt6 is called
312     // Check if timestamp = 20000 (equivalent to 1s)
313     if (++timestamp == 20000)
314     {
315         // set AUX 1 output to high
316         aux_value = 1;
317         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
318 
319         // reset timestamp counter
320         timestamp=0;
321 
322         // Change stimulation pulse
323         WRITE_REGISTER(0x9A80, 0x1000 * seg +  0x100);
324 
325         // Set AUX output low
326         aux_value = 0;
327         WRITE_REGISTER(IFB_AUX_OUT, aux_value); // set AUX 1 to value zero
328 
329         // Increment stimulation pulse memory location
330             seg++;
331 
332         // Stop stimulating after stimulating with all stimulation pulses
333         if(seg == 16)
334             WRITE_REGISTER(0x9A80, 0);
335     }
        ...
411 }
```

In the following section I will discuss the different control algorithms I coded into the system and how they were implemented.

3.3.2 Closed-Loop Control Algorithms

As discussed in the aims of this project, we want to test the performance of different control algorithms on animal models (rats), namely the *on-off* and *dual-threshold amplitude modulation control algorithms* as well as the *proportional* and *proportional-integral amplitude* and *frequency modulation control algorithms*. In this section, I will discuss, in detail, how these algorithms are implemented and in the results section I provide the actual code that was used to implement these control algorithms on the W2100 system.

For all control algorithms, a target level for the beta-band average rectified value (ARV) was defined, such that a beta ARV value above or below that target was considered to be pathological or healthy beta activity, respectively. The input to the controller, e (the error), at time t, is calculated as the normalised error between the measured beta ARV, $b_{measured}$, and the target beta ARV, b_{target}, according to:

$$e(t) = \frac{b_{measured(t)} - b_{target}}{b_{target}}$$

The controller runs every $20ms$ ($T_{controller}$), updating the DBS parameter at each controller call.

For amplitude modulation control algorithms (i.e. on-off, dual-threshold, P and PI amplitude modulation control algorithms), the DBS frequency and pulse duration are fixed at $130Hz$ and $60\mu s$, respectively, with the amplitude varying between $0 - 3mA$. The 16 pulses are set to have amplitudes linearly distributed between $0 - 3mA$, and so the step in DBS amplitude is calculated as $\Delta_{DBS\_amp} = \frac{3mA-0mA}{16-1} = 0.2mA/pulse$.

For the P and PI frequency modulation control algorithms, the DBS amplitude and pulse duration are fixed at $1.5mA$ and $60\mu s$, respectively, with the frequency varying between $0 - 250Hz$. The 16 pulses are set to have frequencies linearly distributed between $0 - 250Hz$, and so the step in DBS frequency is calculated as $\Delta_{DBS\_freq} = \frac{250Hz-0Hz}{16-1} = 16.67Hz/pulse$.

On-Off Control

For the on-off control, there's a single target level for the beta ARV. At time t, depending on whether the error e is positive or negative, the stimulation amplitude, u, is increased towards the upper stimulation amplitude limit, u_{max}, or decreased towards the lower stimulation amplitude limit, u_{min}, respectively. The behaviour of the on-off control algorithm is mathematically described as:

$$u(t) = \begin{cases} u(t-1) + \Delta_{DBS\_amp}; & e(t) > 0 \\ u(t-1) - \Delta_{DBS\_amp}; & e(t) < 0 \end{cases} \tag{3.1}$$

where $u_{min} \leq u(t) \leq u_{max}$, $u_{min} = 0mA$, $u_{max} = 3mA$ and $\Delta_{DBS\_amp} = 0.2mA/pulse$

Dual-Threshold Control

For the dual-threshold control, there's a target range for the beta ARV. The upper and lower bounds of the target range are selected as the 20th and 10th percentiles of the beta ARV when DBS is off. At time t, depending on whether the beta ARV is above the upper bound or below lower bound, the error, e, is calculated with respect to the upper or lower bound, respectively. The stimulation amplitude, u, is increased, decreased or unchanged depending on whether the error is positive, negative or null, respectively. Mathematically, the dual-threshold control algorithm can be described as:

$$u(t) = \begin{cases} u(t-1) + \Delta_{DBS\_amp}; & e(t) > 0 \\ u((t-1); & e(t) = 0 \\ u(t-1) - \Delta_{DBS\_amp}; & e(t) < 0 \end{cases} \tag{3.2}$$

where $u_{min} \leq u(t) \leq u_{max}$, $u_{min} = 0mA$, $u_{max} = 3mA$ and $\Delta_{DBS\_amp} = 0.2mA/pulse$

Proportional-Integral (PI) and Proportional (P) Control

PI and P control algorithms can be implemented as both, frequency and amplitude modulation control algorithms. PI and P control algorithms use a single target level for the beta ARV, with respect to which the error, e, is calculated. For the PI control algorithm, the DBS modulated

parameter, u, being the amplitude or frequency of stimulation, is calculated, at time t, as follows:

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int e(\tau) d\tau \right) \tag{3.3}$$

where K_p is the controller proportional gain and T_i is the controller integral time constant.

Thus, the PI control algorithm incorporates the current and integrated history of the controller measured error, e, at time t. In order to prevent integral wind-up, the integration of the error, e, is paused if the modulated DBS parameter reached its upper or lower parameter bounds. This is known as the condition integration. The P control modulated DBS parameters is calculated using the same formula as that for PI control, but with the integral term being omitted.

The choice of the right controller proportional gain, K_p, and controller integral time constant, T_i, is essential for the success of these control algorithms. The paper published by John Fleming and Professor Madeleine Lowery [22], reported the use of $(K_p, T_i) = (0.23, 0.20)$ for PI amplitude modulation control algorithm and $(K_p, T_i) = (19.30, 0.20)$ for PI frequency modulation control algorithm. Whereas for the P control algorithm, they reported the use of $K_p = 5.0$ and $K_p = 417$ for amplitude and frequency modulation control algorithms, respectively.

3.3.3 W2100 Setup Protocol for DSP-Triggered Stimulation

System Configuration

In the case of DSP-triggered stimulation, one can't use MC Experimenter software to configure the system i.e. to connect the headstage to the PC connected unit. To do so, one needs to use the W2100 Config V1.4.5 software available to download in the following link `https://download.multichannelsystems.com/tmp/jp/W2100_Config-1.4.5.exe`.

In terms of hardware, kindly follow the steps below:

1. Connect the W2100 wireless receiver to the interface board.
2. Connect both ports A and B of the interface board to the PC.
3. Turn on the system.
4. Connect the battery to the headstage and have it very near to the receiver in order to easily detect it.

For further information on the W2100 Config software kindly consult the following link `https://www.multichannelsystems.com/software/w2100-system-config#docs`.

In terms of software, after opening the W2100 Config V1.4.5 software, kindly follow the steps detailed below to configure the system, as shown in figure 3.15:

1. Select the W2100 system connected to the PC from the "Selected W2100 System" drop down mean. Each W2100 system has 2 ports (A and B) and so in the drop down menu, there will be two options to chose from. Kindly chose port B as it is the one through which the DSP firmware is uploaded to the system.

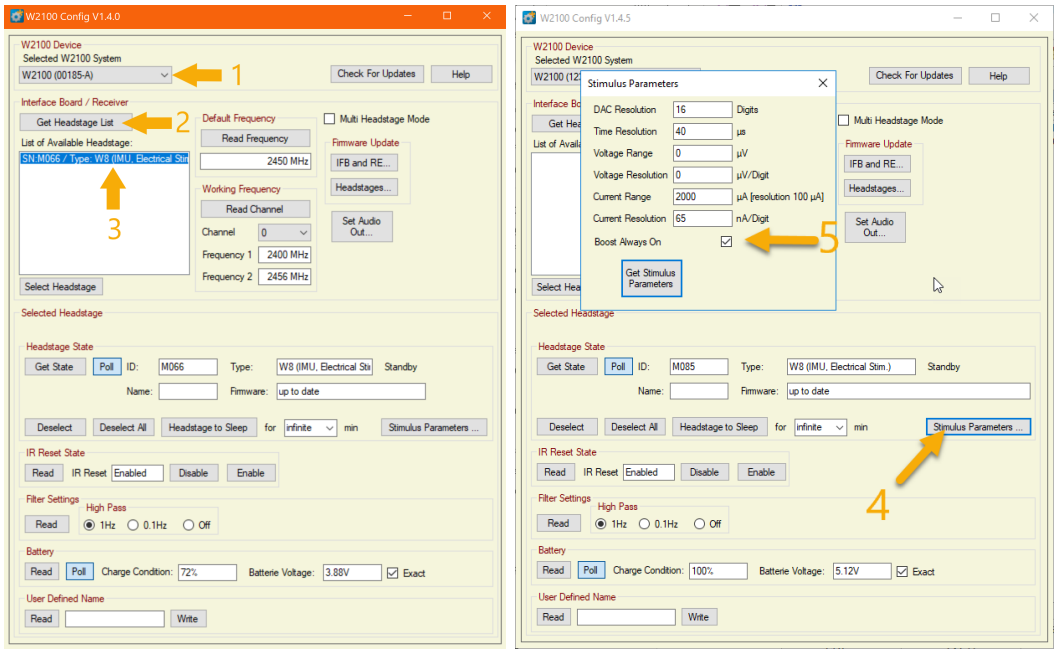
   NB: In figure 3.15 (a) I chose port A because the W2100 Config V1.4.0, I was using when this screenshot was taken, used to upload the DSP firmware using port A. However, the latest W2100 Config V1.4.5 uploads the DSP firmware using port B.

2. Click on "Get Headstage List".

3. Double click on the headstage that will show up in the list of available headstages.

4. Click on "Stimulus Parameters".

5. Enable "Boost Always On" option in the pop-up dialog box. This will have the stimulator always on where it repeats the pulse it just finished applying or changes pulse if asked to.



(a) Steps 1-3 (NB: choose port B not A)

(b) Steps 4-5

Figure 3.15: Steps for setting up W2100 Config Software

Recording Data

Recording data has to be done using MC Experimenter software, as in the case of threshold-triggered stimulation. NB: For the threshold triggered stimulation, I used MC Experimenter for configuring the system and recording data. However, for the DSP-triggered stimulation, MC Experimenter is used just for recording data.

After being done with setting up the system using W2100 config V1.4.5, open MC Experimenter and follow the steps detailed below, as shown in figure 3.16:

1. Drag the "W2100 A" and "Recorder" instruments from the main menu.
2. Connect the blue and green output ports of the "W2100" instrument into the recorder input port to record the headstage recorded and analog inputs data respectively.
3. Click on "Start DAQ" to start streaming data into the MC Experimenter GUI.
4. Click on "Recording off" to start recording data.

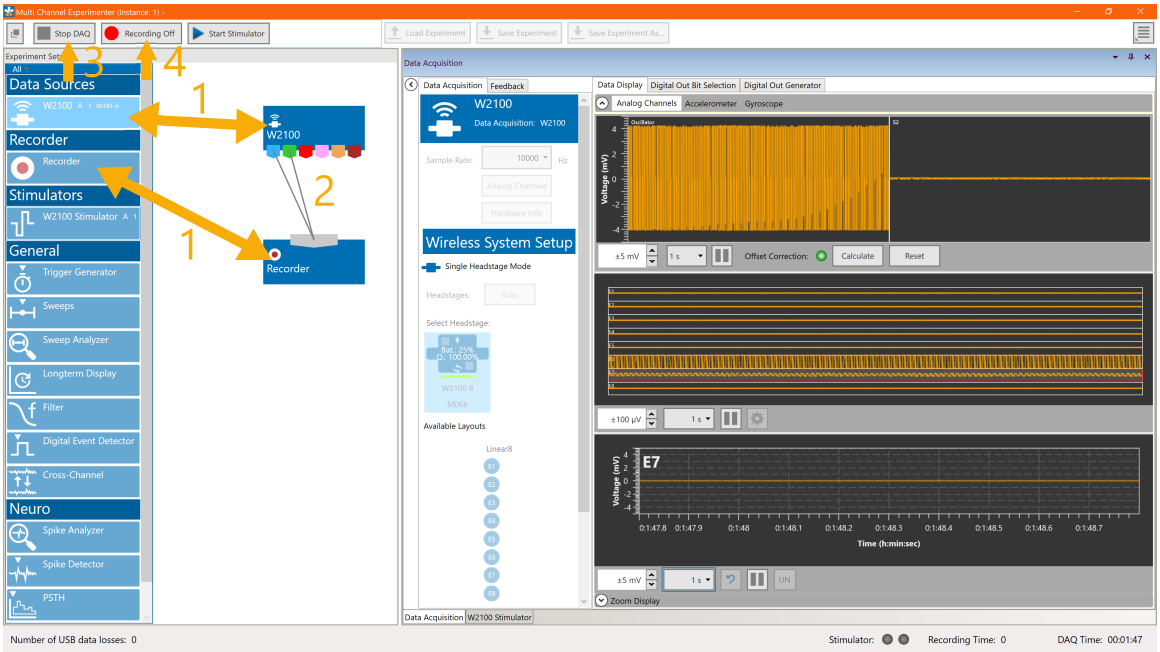


Figure 3.16: MC Experimenter Setup for Signal Recording

For further information on how to use MC Experimenter software, kindly consult the following link `https://www.multichannelsystems.com/software/multi-channel-experimenter#docs`. Once the data has been recorded, it can be exported into CVS format for offline processing in Matlab and Python using MC DataManager software as detailed in the appendix 7.1.

DSP Code Compilation

Compiling the DSP code is done through Code Composer Studio IDE V8 from Texas Instruments which can be downloaded through the following link `http://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html`. Once downloaded, install version 7.4.x of the C6000 compiler (as follows: Menu ->Help ->Install Code Generation Compiler Tools...; TI Compiler Updates).

Once the closed-loop control algorithm has been coded into the DSP firmware as discussed in 3.3.1, the next step would be to compile that code into a binary file (`McsDspRealtimeFeedback\DSP\FB_W2100_SCU_MEA256\ReleaseFB\_W2100\_SCU\_MEA256.bin`) that will be uploaded later on into the DSP.

To compile the DSP firmware, once Code Composer Studio IDE V8 is opened, follow the steps detailed below, as shown in figures 3.17, 3.18 and 3.19:

1. Click on "Import Project".
2. In the pop-up dialog box, click on "Browse"and choose the "FB_W2100_SCU_MEA256" folder of the following directory `McsDspRealtimeFeedback\DSP\FB_W2100_SCU_MEA256` which will show up in the list of discovered projects.
3. Click on "Finish" .

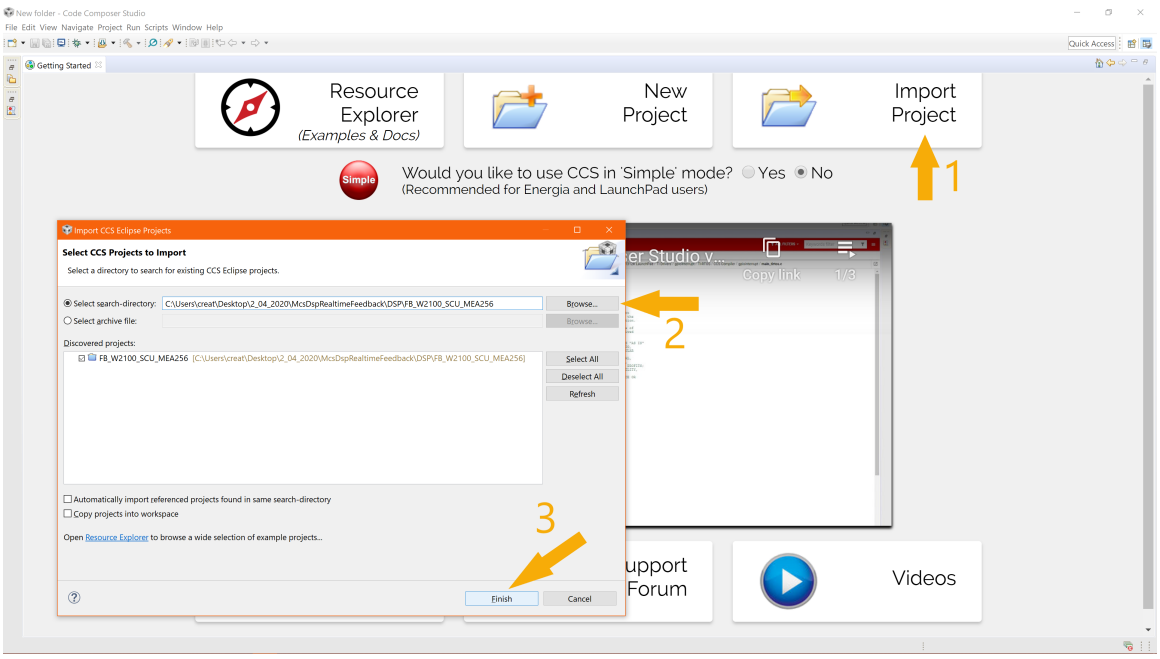


Figure 3.17: Steps 1-3 for Compiling DSP Firmware

4. Set the Code Composer Studio IDE to release mode as follows: Path ->Build Configuration ->Set Active ->Release, as shown in figure 3.18

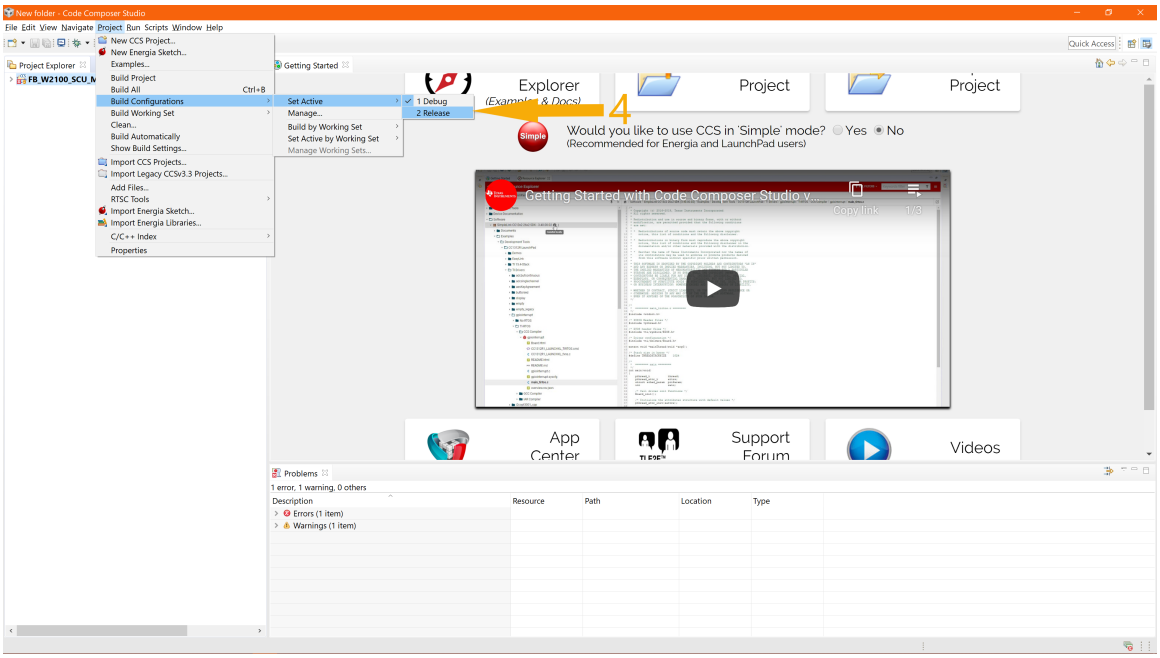


Figure 3.18: Step 4 for Compiling DSP Firmware

5. Click on the "Build" icon. If the compilation process was successful, "Build Finished" will pop up in the Console as shown in figure 3.19.

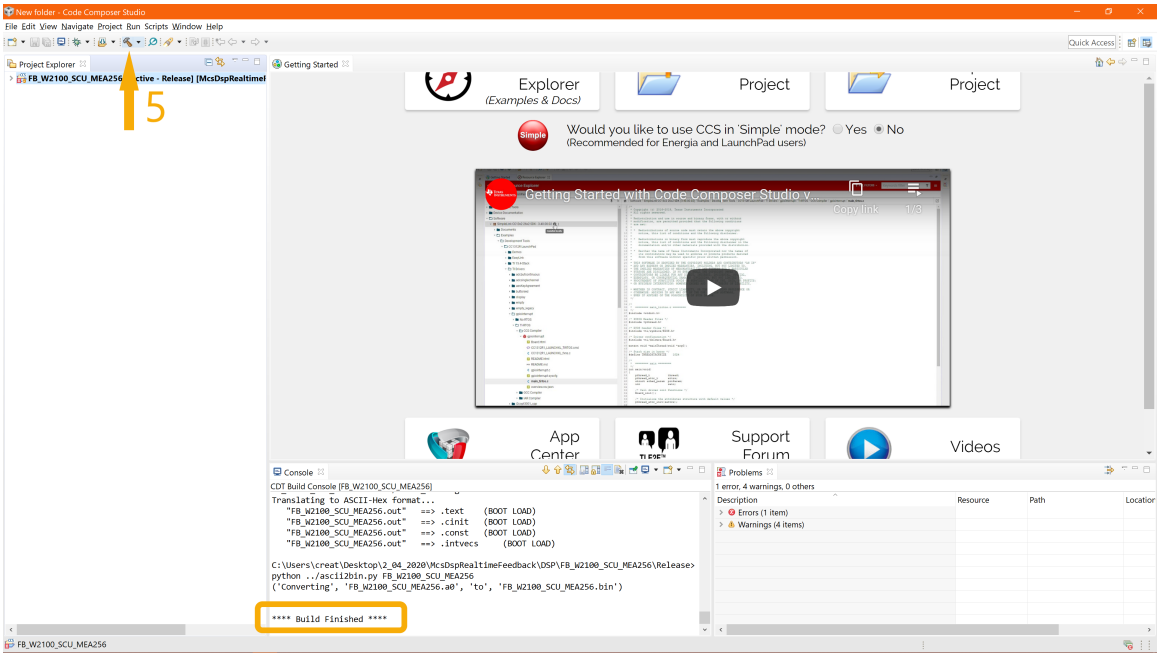


Figure 3.19: Step 5 for Compiling DSP Firmware

Pulses Patterns Upload and Data Acquisition

Compiling the `McsDspRealtimeFeedback\Windows\DSPDacqExample\DSPDacqExample.csproj` project uploads the pulse patterns into the headstage and starts data acquisition and streaming. To do so, kindly follows the steps detailed below, as shown in figures 3.20, 3.21 and 3.22:

1. After opening Visual Studio 2019, click on "Open a project or solution" tab.

2. In the pop up dialog box, choose "DSP Dacq Example.csproj" located in `McsDspRealtimeFeedback\Windows\DSPDacqExample`.

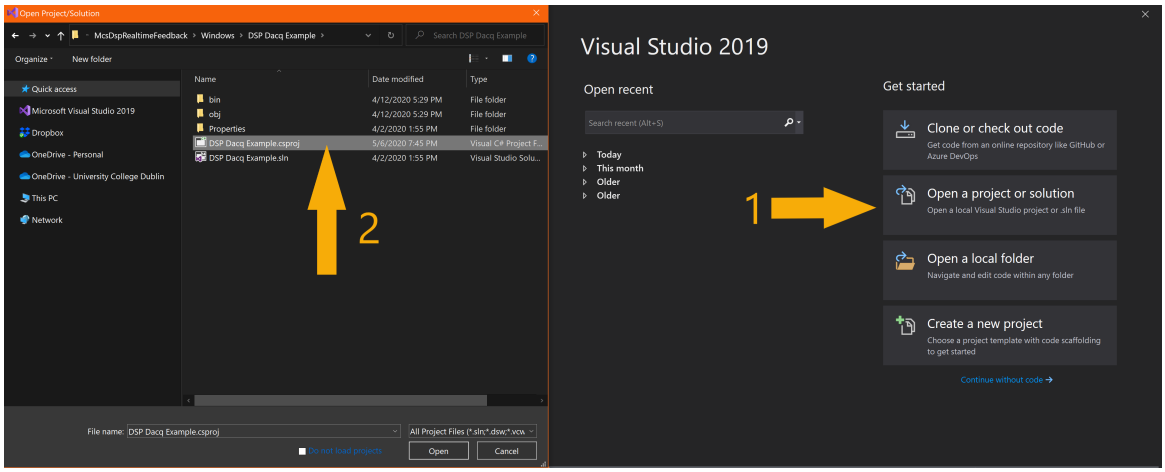


Figure 3.20: Steps 1-2 for uploading Stimulation Pulses and Turning on Data Acquisition

3. Click on "start" icon.

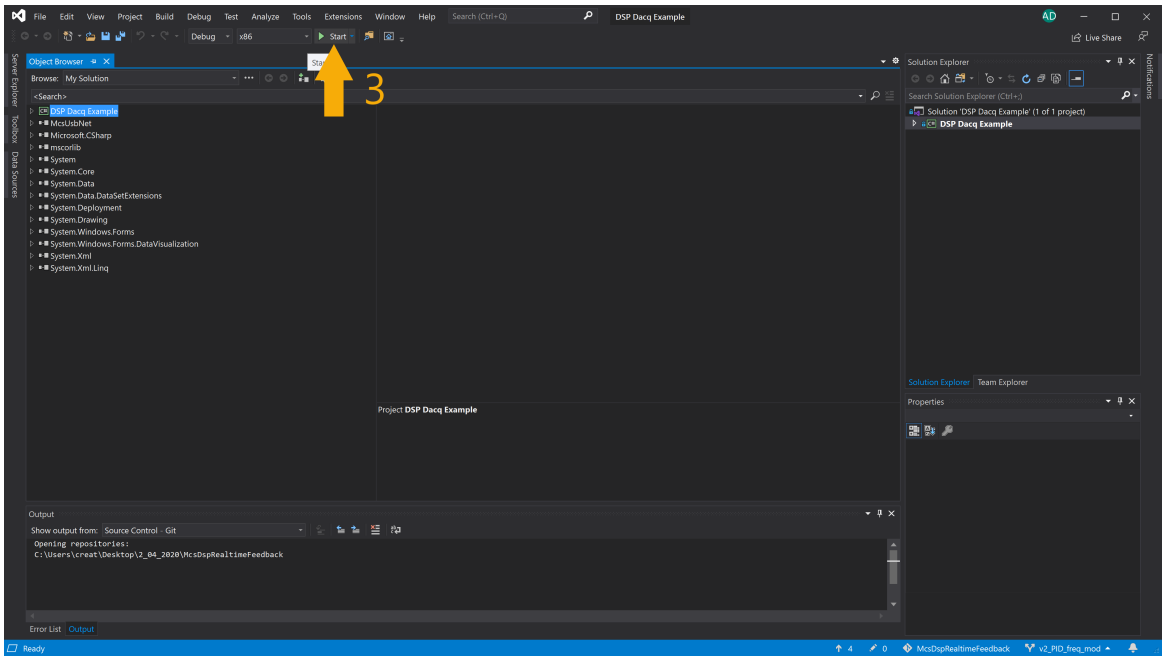


Figure 3.21: Step 3 for uploading Stimulation Pulses and Turning on Data Acquisition

4. In the pop up dialog box, choose "W2100-B" from the "MCS Devices" drop down menu. NB: In figure 3.22 I chose "W2100-A" device because the old firmware used port A to upload the pulses to the W2100 system. However, for the current, new firmware version, this is done through port B.

5. Click on "Start Measurement". This will turn on data acquisition from the headstage recording electrodes and the analog and digital inputs to the interface board.

6. Choose which data to plot from the drop down menu associated with each of the 4 charts. One could plot the data from the recording electrodes, analog and digital inputs to the interface board, and the 64 MonitorData variables declared in the DSP code, as previously discussed.

   NB: In the drop down, the MonitorData variables are labelled as "DSP" instead of "MonitorData".

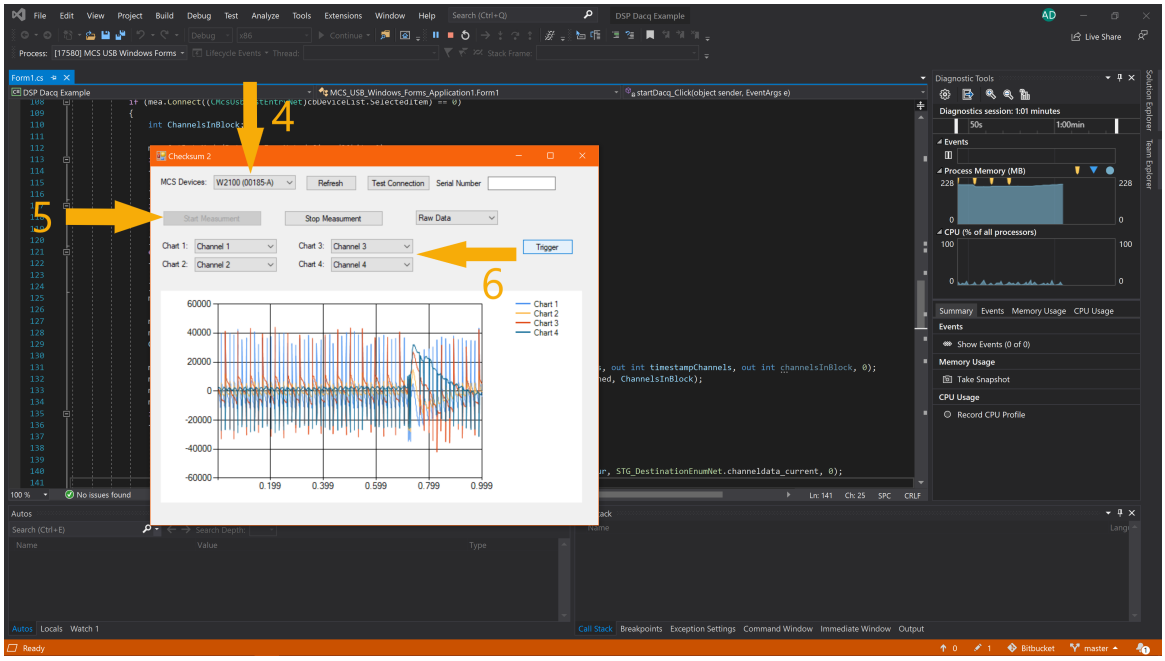


Figure 3.22: Steps 4-6 for uploading Stimulation Pulses and Turning on Data Acquisition

Uploading DSP Firmware

Compiling the `McsDspRealtimeFeedback\Windows\FB_Example\"FB\_Example.csproj` project uploads the DSP firmware into the W2100 system to enable closed-loop control. To do so, kindly follows the steps detailed below, as shown in figures 3.23, 3.24 and 3.25:

1. After opening Visual Studio 2019, click on "Open a project or solution" tab.

2. In the pop up dialog box, choose "FB_Example.csproj" project located in `McsDspRealtimeFeedback\Windows\FB_Example`.

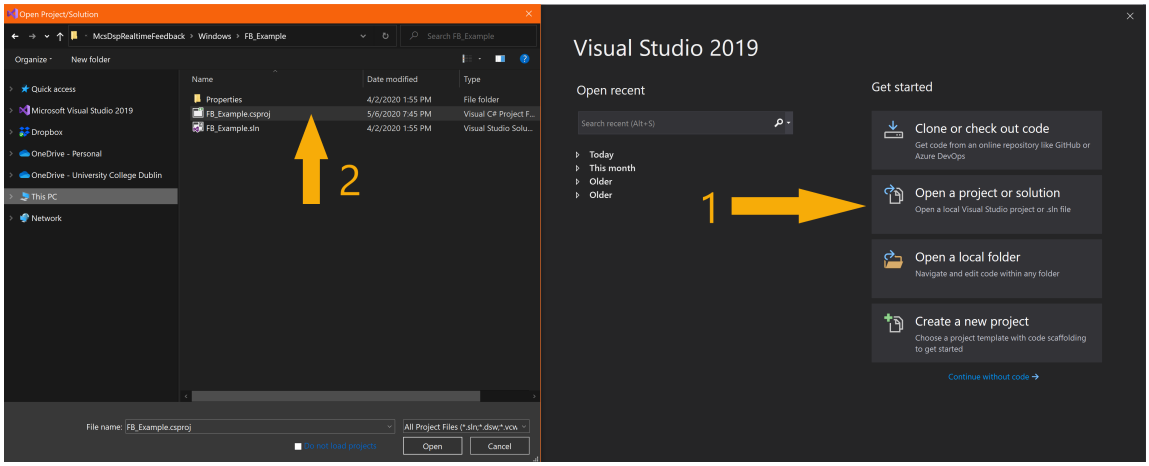


Figure 3.23: Steps 1-2 for Uploading DSP Firmware

3. Click on "start" icon.

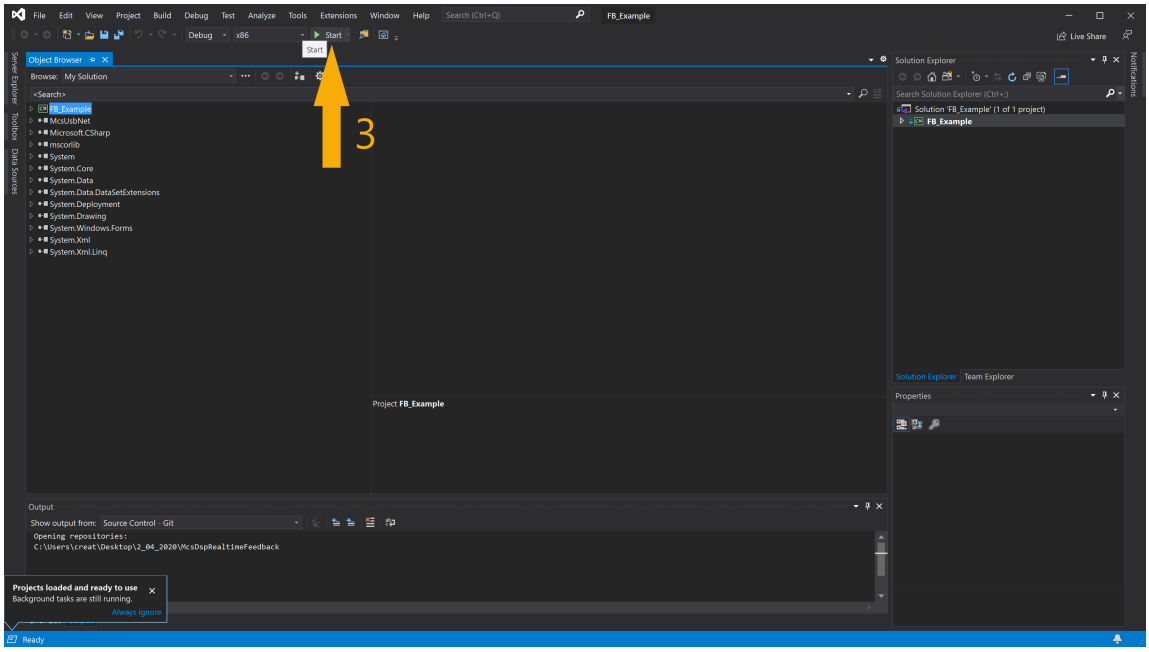


Figure 3.24: Step 3 for Uploading DSP Firmware

4. In the pop up dialog box, click on "Upload DSP Binary". To turn off the DSP, click on "Stop DSP".

   NB: All other components of the dialog box do not work, they are just there as a template to be used if needed in the future.

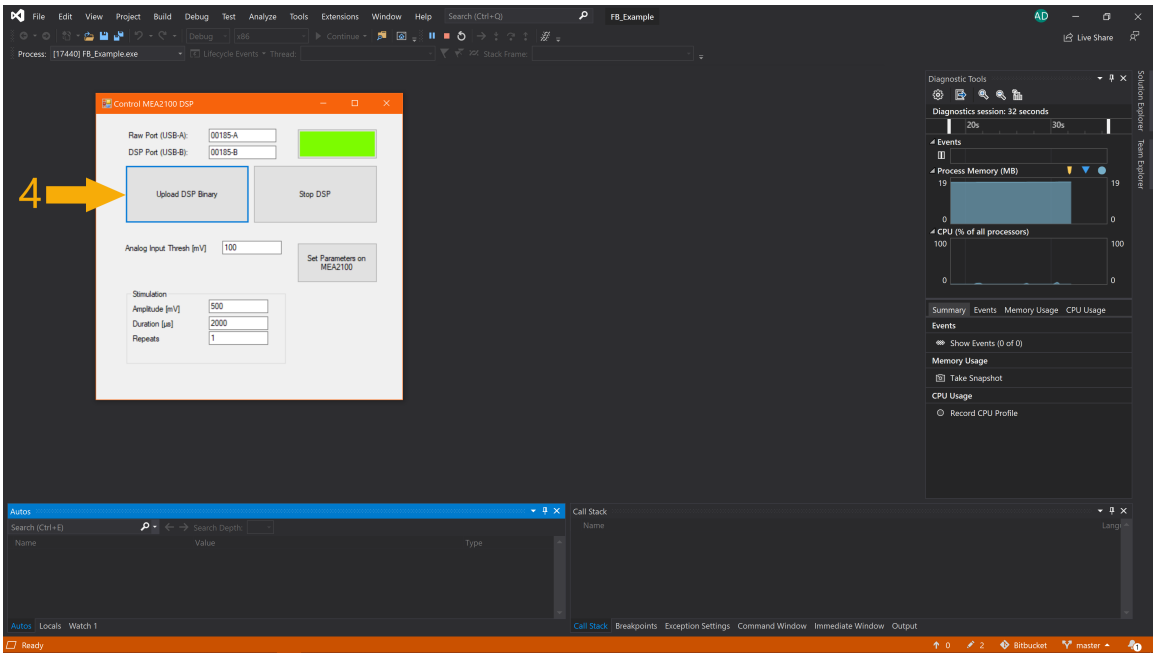


Figure 3.25: Step 4 for Uploading DSP Firmware. NB: The green box indicates that the W2100 system is connected to the PC , otherwise it would be red.

Chapter 4: **Results and Discussion**

4.1 Threshold-Triggered Stimulation Delays

Figure 4.1 shows the recorded signal from electrode 6 (connected to the external signal generator), the analog input to the IFB recording the voltage across the external signal generator (Analog 1) and the data from the "Trigger" output showing when the feedback stimulation was triggered.

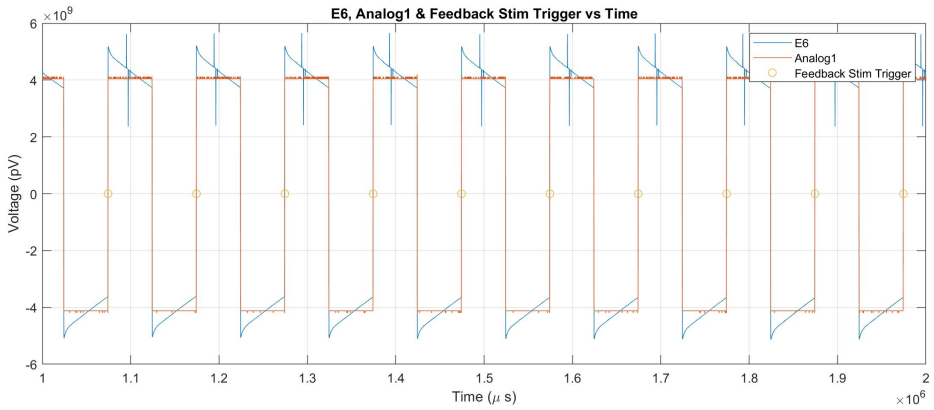


Figure 4.1: Headstage Electrode 6 and Analog Input 1 (to the IFB) Signals and Stimulator Trigger Time

Figure 4.2 provides zoomed in view of figure 4.1.

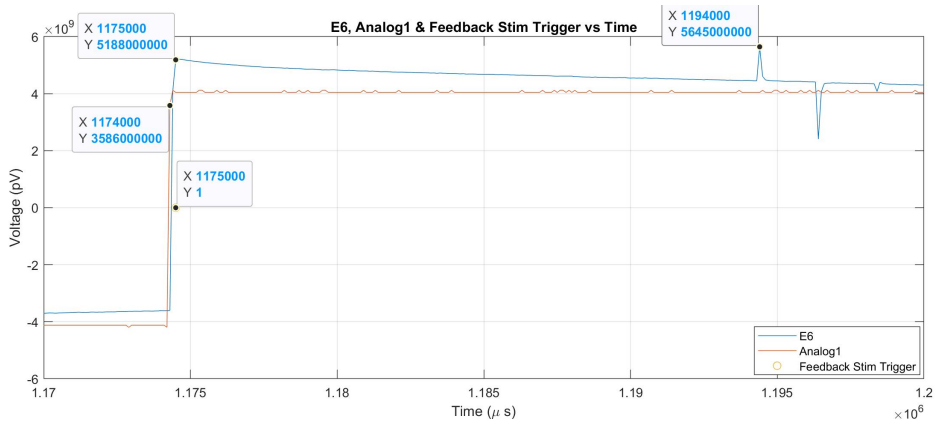


Figure 4.2: Zoomed In Headstage Electrode 6 and Analog Input 1 (to the IFB) Signals and Stimulator Trigger Time

Recording Delay: comparing the recorded signal from the headstage with that of the the analog input, the recording delay is quantified as being $1ms$.

DSP Delay: comparing when the recorded signal went high i.e. when the feedback was supposed to be triggered with when the feedback was triggered, the signal processing delay is quantified as being $0ms$.

Stimulation Delay: comparing when the feedback was triggered and when I saw an output at the stimulating electrode picked by the recording electrode, the stimulation delay is quantified as being $19ms$. This delays includes the stimulation delay and also the recording delay. Since the recording delay has been previously quantified as $1ms$, then the stimulation delay is $18ms$.

Threshold-Triggered Feedback Stimulation Total Delay = Recording Delay + DSP Delay + Stimulation Delay = 19ms

4.2 DSP-Triggered Stimulation Code

As discussed previously, the code for implementing closed-loop stimulation is split into two segments, the first being the code through which the different 16 pulses are defined, and the second being the code associated with implementing the control algorithm itself. I will discuss each of these two sections separately. Kindly find in the following link `https://github.com/adildahlan/McsDspRealtimeFeedback`, the repository to the different system's firmware/code versions associated with implementing each of the different control algorithms.

4.2.1 Pulses Declaration Code

Case 1: Amplitude Modulation Control Algorithms

As discussed in section 3.3.2, for all amplitude modulation algorithms, namely the on-off control, dual-threshold control, P and PI amplitude modulation control algorithms, the 16 pulses are defined such that the DBS frequency and pulse duration are fixed at $130Hz$ and $60\mu s$, respectively. The pulses are set to have amplitudes linearly distributed between $0mA$ and $3mA$. Thus, all

amplitude modulation control algorithms have the same pulses and so the same code associated with defining them. Below is the section of the `startDacq_Click` function associated with defining the 16 pulses of the amplitude modulation control algorithms:

```
143 private void startDacq_Click(object sender, EventArgs e)
144 {
145     // Define stimulation frequency and period
146     int f_stim = 130;
147     int T_stim = (int)(1 / f_stim * 10^6 );                    //in us
148 
149     // Define on and off phases of stimulation pulse           (in us)
150     int pulse_on_phase_dur = 60;
151     int pulse_off_phase_dur = T_stim - 2 * pulse_on_phase_dur;
152 
153     // Define the upper and lower bounds for the controller output
154     const int MaxValue = 3000;                                 //in uA
155     const int MinValue = 0;                                    //in uA
156 
157     // Define the step between pulses amplitude
158     const int delta_DBS_amp = (MaxValue - MinValue) / 15;    // in uA
        ...
228     // Define the amplitude vector of the 3 segments of the biphasic pulse (in uA)
229     int[] ampl = new[] { 1000000, -1000000, 0 };
230 
231     // Define the duraion vector of the 3 segments of the biphasic pulse   (in us)
232     ulong[] dur = new ulong[] { 2000, 2000, 10000 - 2 * 2000 };
233     // Define each pulse
234     for (int i = 0; i < 16; i++)
235     {
236         // Define the amplitude (uA) of each of the 3 segments
237         ampl[0] = -i * delta_DBS_amp;
238         ampl[1] = i * delta_DBS_amp;
239         // Define the duration (us) of each of the 3 segments
240         dur[0] = (ulong)pulse_on_phase_dur;
241         dur[1] = (ulong)pulse_on_phase_dur;
242         dur[2] = (ulong)pulse_off_phase_dur;
243 
244         // Define the associated pulse
245         CStimulusFunctionNet.StimulusDeviceDataAndUnrolledData prep = stim.PrepareData(0,
            ampl, dur, STG_DestinationEnumNet.channeldata_current, 1);
246 
247         // Check the available memory in the headstage
248         if (first)
249         {
250             first = false;
251             preplegth = prep.DeviceDataLength;
252         }
253         // Check that the pulse fits into the designated memory
254         Debug.Assert(preplegth == prep.DeviceDataLength);
255         Debug.Assert(prep.DeviceDataLength <= 15);
256         // Store pulse into designated memory
257         stim.SendPreparedData(0x10 * i + 0, prep,
            STG_DestinationEnumNet.channeldata_current);
258     }
        ...
279 }
```

Case 2: Frequency Modulation Control Algorithms

As discussed in section 3.3.2, for all frequency modulation algorithms, namely the P and PI frequency modulation control algorithms, the 16 pulses are defined such that the DBS amplitude and pulse duration are fixed at $1.5mA$ and $60\mu s$, respectively and the frequencies are linearly distributed between $0Hz$ and $250Hz$. Thus, all frequency modulation control algorithms have the same pulses (which only vary in the duration of the off-phase) and so they have the same code associated with defining them. Below is the section of the `startDacq_Click` function associated with defining the 16 pulses of the frequency modulation control algorithms:

```
143 private void startDacq_Click(object sender, EventArgs e)
144 {
145     // Define the upper and lower bounds for the controller output (i.e. DBS frequency)
146     const int MaxValue = 250;                                   //in Hz
147     const int MinValue = 0;                                     //in Hz
148
149     // Define the step between pulses frequency
150     const int delta_DBS_freq = (MaxValue - MinValue) / 15; //in Hz
151     // Define on phase of stimulation pulse
152     const int pulse_on_phase_dur = 60;                          //in us
153
154     // Define stimulation pulse amplitude
155     const int stim_amp = 1500;                                  //in uA
156
157     // Define variable to store stimulation frequency
158     int f_stim = 0;                                             //in Hz
        ...
228     // Define the amplitude vector of the 3 segments of the pulse (in uA) (random values)
229     int[] ampl = new[] { 1000000, -1000000, 0 };
230
231     // Define the duration vector of the 3 segments of the pulse  (in us) (random values)
232     ulong[] dur = new ulong[] { 2000, 2000, 10000 - 2 * 2000 };
233
234     // Define each pulse
235     for (int i = 0; i < 16; i++)
236     {
237         // The case of i == 0 is associated with no pulse, thus we treat it separately,
238         // by setting the pulse amplitude to be zero and DBS frequency to be that of i==1
239         if (i == 0)
240         {
241             f_stim = delta_DBS_freq;
242             ampl[0] = 0;
243             ampl[1] = 0;
244         }
245         // for all i != 0, DBS is on
246         else
247         {
248             // Calculate stimulation frequency
249             f_stim = i * delta_DBS_freq;
250             // Define the amplitude (uA) of each of the 3 segments
251             ampl[0] = -1 * stim_amp;
252             ampl[1] = stim_amp;
253         }
254         // Define the duration (us) of each of the 3 segments
```

```
255         dur[0] = (ulong)pulse_on_phase_dur;
256         dur[1] = (ulong)pulse_on_phase_dur;
257         dur[2] = (ulong)(1 / f_stim * 10^(6) - 2 * pulse_on_phase_dur);
258
259         // Define the associated pulse
260         CStimulusFunctionNet.StimulusDeviceDataAndUnrolledData prep = stim.PrepareData(0,
            ampl, dur, STG_DestinationEnumNet.channeldata_current, 1);
261
262         // Check the available memory in the headstage
263         if (first)
264         {
265             first = false;
266             preplegth = prep.DeviceDataLength;
267         }
268         // Check that the pulse fits into the designated memory
269         Debug.Assert(preplegth == prep.DeviceDataLength);
270         Debug.Assert(prep.DeviceDataLength <= 15);
271
272         // Store pulse into designated memory
273         stim.SendPreparedData(0x10 * i + 0, prep,
            STG_DestinationEnumNet.channeldata_current);
274     }
        ...
293 }
```

4.2.2 Control Algorithms Code

As discussed in 3.3.1, the control algorithm is coded into the `interrupt6` function which is called each time a new data frame is available at the DSP. The code associated with all control algorithms could be split into several sections which as detailed below:

1. Since the `interrupt6` function is called each time a new frame is available (i.e. the sampling period, T_s) but we want to call the controller at the controller period $T_{controller}$, we need to determine the ratio between $T_{controller}$ and T_s to know every how many calls of the `interrupt6` function we want to call the controller.

2. Defining a vector to store the amplitude or frequencies of the DBS pulses for the amplitude or frequency modulation control algorithms, respectively.

3. Defining the constants and variables associated with implementing the control algorithm.

4. Incrementing the counter for the `interrupt6` function call and access the controller call if-statement when it is equal to the ratio between $T_{controller}$ and T_s.

5. Setting the AUX1 port high at the beginning of the controller call if statement.

6. Calculating the modulated DBS parameter i.e. which pulse amplitude / frequency to apply if we were able to implement the control algorithms in the continuous form.

7. Determining which pulse of the 16 ones is closest to the required one.

8. Changing the stimulation pulse accordingly by writing to the headstage registers.

9. Setting the AUX1 port low at the end of the controller call if statement.

-**Recording Delay:** This is expected to be similar to that in the case of threshold triggered stimulation, namely $1ms$.

-**Algorithm-Processing Delay:** By comparing the time difference between when the AUX1 goes high and low, we could determine the algorithm-processing delay. However, Dr Jens, the Head of Firmware at MultiChannel Systems, told us that it would be negligible, less than the sampling period.

-**Stimulation Delay:** By comparing the time difference between when the AUX1 went low and when the stimulation pulse was changed, as recorded from the stimulation electrode, we could determine the stimulation delay.

To allow the synchronisation of the AUX1 and stimulation electrode signals, we need to record them using the same oscilloscope or set them as as analog inputs into the Interface board of the W2100 system, as previously done when measuring the delays associated with the threshold-triggered stimulation.

On-Off Control

Below is the section of the `interrupt6` function associated with the on-off control algorithm.

```
111 interrupt void interrupt6(void)
112 {
113     // Define sampling frequency and period
114     const float f_s = 20000;
115     const float T_s = 1 / f_s;
116 
117     // Define controller call period (20ms)
118     const float T_controller = 0.02;
119 
```

```
120     // Calculate T_controller to T_s ratio to know every how many calls of interrupt 6 we
        need to call the controller
121     const int ratio_T_controller_T_s = T_controller / T_s;
122
123     // Define SetPoint being the target beta ARV
124     const float SetPoint = 1;                       //set SetPoint randomly to be 1V
125
126     // Define a variable that is true just the first run
127     static int first_run = 1;
128
129     // Define the upper and lower bounds for the controller output (i.e. DBS amplitude)
130     const float MaxValue = 3000 * 10^(-6);          //in A
131     const float MinValue = 0;                       //in A
132
133     // Define the step between pulses amplitude
134     const float delta_DBS_amp = ( MaxValue - MinValue ) / 15;
135
136     // Define vector to store the amplitude of the stimulation pulses
137     static float pulse[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
138     // Define dummy variable
139     static int c = 0;
140     // Multiply pulse vector by delta_DBS_amp to get the vector associated with the
        amplitude of the stimulation pulses
141     if (first_run)
142     {   for (c = 0; c < 16; c++)
143         {
144             pulse[c] = pulse[c] * delta_DBS_amp;
145         }
146         // Not anymore the first run (thus, this if statement will be run only in the first
            call of interrupt6 function)
147         first_run = 0;
148     }
149     // Define current error
150     static float error = 0;
151
152     // Define the applied bounded controller output value (i.e. applied pulse amplitude)
153     static float OutputValue = 0;
154
155     // Set increment initially to 0
156     static float increment = 0.0;
157
158     // Define index of stimulation pulse to be applied
159     static int stim_index = 0;
160
161     // Define difference between the amplitude of the required stimulation pulse and the one
        to be applied
162     static float pulse_amp_diff = 0;
163
164     // Define state value
165     static float state_value = 0;
166
167     // Define counter for function run
168     static int timestamp = 0;
        ...
310     // Increment timestamp each function call and call controller when T_controller elapsed
        i.e. when timestamp == ratio_T_controller_T_s
311     if (++timestamp == ratio_T_controller_T_s)
312     {
313         // set AUX output to 1
314         aux_value &= 1;
315         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
316
317         // reset timestamp counter
318         timestamp=0;
```

```
        // Calculate current beta ARV (differential recording)
        state_value = abs(HS1_Data_p[0] - HS1_Data_p[1]) * 10^(-12); //convert from pV to V

        // Calculate Error - if SetPoint > 0.0,
        // then normalize error with respect to SetPoint
        if (SetPoint==0)
        {
            error = state_value - SetPoint;                    //in V
            increment = 0.0;
        }
        else
        {
            error = (state_value - SetPoint) / SetPoint;          //in V
            if (error>0)
                increment = delta_DBS_amp;
            else
                increment = -1*delta_DBS_amp;
        }
        // Bound the controller output (between MinValue - MaxValue)
        if ( OutputValue + increment > MaxValue )
            OutputValue = MaxValue;
        else if ( OutputValue + increment < MinValue )
            OutputValue = MinValue;
        else
            OutputValue = OutputValue + increment;

        // Determine the pulse closest to OutputValue (and its index)
        // Set randomly the index of the pulse closest to OutputValue to be 0
        stim_index = 0;

        // Calculate the difference between OutputValue and pulse of index stim_index
        pulse_amp_diff = abs(pulse[stim_index] - OutputValue);

        // Pick the stimulation pulse of amplitude closest to OutputValue
        // Loop around all 16 pulses
        for (c = 1; c < 16; c++)
        {
            // Check if this pulse is the closest to OuputValue
            if ( abs(pulse[c] - OutputValue) < pulse_amp_diff)
            {
                // Update the index of the closest pulse to OutputValue
                stim_index = c;
                // Update the difference between OutputValue and pulse of index stim_index
                pulse_amp_diff = abs(pulse[c] - OutputValue);
            }
        }
        // Relate the pulse index to the memory segment index associated with it
        seg = stim_index;

        // Update stimulation pulse
        WRITE_REGISTER(0x9A80, 0x1000 * seg +  0x100);

        // Set AUX 1 output value to zero
        aux_value &= ~1;
        WRITE_REGISTER(IFB_AUX_OUT, aux_value);
     }
    ...
}
```

Dual-Threshold Control

Below is the section of the `interrupt6` function associated with the dual-threshold control algorithm.

```
111 interrupt void interrupt6(void)
112 {
113     // Define sampling frequency and period
114     const float f_s = 20000;
115     const float T_s = 1 / f_s;
116 
117     // Define controller call period (20ms)
118     const float T_controller = 0.02;
119 
120     // Calculate T_controller to T_s ratio to know every how many calls of interrupt 6 we
        need to call the controller
121     const int ratio_T_controller_T_s = T_controller / T_s;
122 
123     // Define the upper threshold for target beta ARV range
124     const float UpperThreshold = 1;           // Set to 1V (random value)
125 
126     // Define the lower threshold for target beta ARV range
127     const float LowerThreshold = 0.5;        // Set to 0.5V (random value)
128 
129     // Define a variable that is true just the first run
130     static int first_run = 1;
131 
132     // Define the upper and lower bounds for the controller output (i.e. DBS amplitude)
133     const float MaxValue = 3000 * 10^(-6);          //in A
134     const float MinValue = 0;                       //in A
135 
136     // Define the step between pulses amplitude
137     const float delta_DBS_amp = ( MaxValue - MinValue ) / 15;
138 
139     // Define vector to store the amplitude of the stimulation pulses
140     static float pulse[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
141 
142     // Define dummy variable
143     static int c = 0;
144 
145     // Multiply pulse vector by delta_DBS_amp to get the vector associated with the
        amplitude of the stimulation pulses
146     if (first_run)
147     {
148         for (c = 0; c < 16; c++)
149         {
150             pulse[c] = pulse[c] * delta_DBS_amp;
151         }
152 
153         // Not anymore the first run (thus, this if statement will be run only in the first
            call of interrupt6 function)
154         first_run = 0;
155     }
156 
157     // Define current error
158     static float error = 0.0;
159 
160     // Define the applied bounded controller output value (i.e. applied pulse amplitude)
161     static float OutputValue = 0.0;
162 
163     // Set increment initially to 0
```

```
164     static float increment = 0.0;
165
166     // Define index of stimulation pulse to be applied
167     static int stim_index = 0;
168
169     // Define difference between the amplitude of the required stimulation pulse and the one
        to be applied
170     static float pulse_amp_diff = 0;
171
172     // Define state value
173     static float state_value = 0;
174
175     // Define counter for function run
176     static int timestamp = 0;
        ...
313     // Increment timestamp each function call and call controller when T_controller elapsed
        i.e. when timestamp == ratio_T_controller_T_s
314     if (++timestamp == ratio_T_controller_T_s)
315     {
316         // set AUX output to 1
317         aux_value &= 1;
318         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
319
320         // reset timestamp counter
321         timestamp=0;
322         // Calculate current beta ARV (differential recording)
323         state_value = abs(HS1_Data_p[0] - HS1_Data_p[1]) * 10^(-12); //convert from pV to V
324
325         // Calculate error with respect to upper/lower threshold
326         if (state_value > UpperThreshold)           // Increase if above upper threshold
327         {
328             error = (state_value - UpperThreshold) / UpperThreshold;
329             increment = delta_DBS_amp;
330         }
331         else if (state_value < LowerThreshold)      // Decrease if below lower threshold
332         {
333             error = (state_value - LowerThreshold) / LowerThreshold;
334             increment = -delta_DBS_amp;
335         }
336         else                    // Do nothing when within upper and lower thresholds
337         {
338             error = 0;
339             increment = 0;
340         }
341
342         // Bound the controller output (between MinValue - MaxValue)
343         if ( OutputValue + increment > MaxValue )
344             OutputValue = MaxValue;
345         else if ( OutputValue + increment < MinValue )
346             OutputValue = MinValue;
347         else
348             OutputValue = OutputValue + increment;
349
350         // Determine the pulse closest to OutputValue (and its index)
351         // Set randomly the index of the pulse closest to OutputValue to be 0
352         stim_index = 0;
353
354         // Calculate the difference between OutputValue and pulse of index stim_index
355         pulse_amp_diff = abs(pulse[stim_index] - OutputValue);
356
357         // Loop around all 16 pulses to determine the one closest to OutputValue
358         for (c = 1; c < 16; c++)
359         {
360             // Check if this pulse is the closest to OuputValue
```

```
            if ( abs(pulse[c] - OutputValue) < pulse_amp_diff )
            {
                // Update the index of the closest pulse to OutputValue
                stim_index = c;
                // Update the difference between OutputValue and pulse of index stim_index
                pulse_amp_diff = abs(pulse[c] - OutputValue);
            }
        }

        // Relate the pulse index to the memory segment index assoicated with it
        seg = stim_index;

        // Update stimulation pulse
        WRITE_REGISTER(0x9A80, 0x1000 * seg +  0x100); // Trigger Channel 1

        // Set AUX 1 output value to zero
        aux_value &= ~1;
        WRITE_REGISTER(IFB_AUX_OUT, aux_value);
     }
    ...
}
```

P and PI Amplitude and Frequency Modulation

As discussed in 3.3.2, the P and PI amplitude and frequency modulation control algorithms are very similar in terms of calculating the modulated DBS parameter (DBS amplitude or frequency) which is given by formula 3.3, except for the fact that the integral term is not included in the P control algorithm and for the fact that each of these algorithms has its own value fr the controller proportional gain, K_p, and controller integral time constant, T_i as summarized in the table below

Table 4.1: P and PI Amplitude and Frequency Modulation Controller Parameters

Control Algorithm	**Proportional Gain** (K_p)	**Integral Time Constant** (T_i)
PI Amplitude Modulation	0.23	0.20
PI Frequency Modulation	19.30	0.20
P Amplitude Modulation	5	NA
P Frequency Modulation	417	NA

Thus, the code for implementing all these 4 algorithms is very similar and so I will provide below the section of the `interrupt6` function associated with implementing the PI frequency modulation

control algorithm. At the end, I will discuss the changes to be made in order to implement PI amplitude modulation and the P amplitude and frequency modulation control algorithms whose code is provided in appendices 7.2, 7.3 and 7.4, respectively.

```
111 interrupt void interrupt6(void)
112 {
113     // Define sampling frequency and period
114     const float f_s = 20000;
115     const float T_s = 1 / f_s;
116 
117     // Define algorithm period  (20ms)
118     const float T_controller = 0.02;
119 
120     // Calculate T_controller to T_s ratio to know every how many calls of interrupt 6 we
        need to call the controller
121     const int ratio_T_controller_T_s = T_controller / T_s;
122 
123     // Define SetPoint being the target beta ARV
124     const float SetPoint = 1;                       //set SetPoint randomly to be 1V
125 
126     // Determine how aggressively the controller reacts to the current error with setting
        Proportional Gain
127     const float Kp = 19.30;                          //in AU
128 
129     // Determine how fast the controller integrates the error history by setting Integral
        Time Constant
130     const float Ti = 0.20;                          //in s
131 
132     // Define a variable that is true just the first run
133     static int first_run = 1;
134 
135     // Define the upper and lower bounds for the controller output (i.e. DBS frequency)
136     const float MaxValue = 250;                     //in Hz
137     const float MinValue = 0;                       //in Hz
138 
139     // Define the step between pulses frequency
140     const float delta_DBS_freq = (MaxValue - MinValue) / 15;
141 
142     // Define vector to store the frequency of the stimulation pulses
143     static float pulse[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
144 
145     // Define dummy variable
146     static int c = 0;
147 
148     // Multiply pulse vector by delta_DBS_freq to get the vector associated with the
        frequency of the stimulation pulses
149     if (first_run)
150     {
151         for (c = 0; c < 16; c++)
152         {
153             pulse[c] = pulse[c] * delta_DBS_freq;
154         }
155         // Not anymore the first run (thus, this if statement will be run only in the first
            call of interrupt6 function)
156         first_run = 0;
157     }
158 
159     // Define integral term
160     static float ITerm = 0;
161 
162     // Define difference in time between previous and current frames
```

```
163     const float delta_time = T_s;
164
165     // Define current and last errors
166     static float error = 0;
167
168     // Define u(t) which is calculated controller output (i.e. pulse frequency)
169     static float u = 0;
170
171     // Define the applied bounded controller output value (i.e. applied pulse frequency)
172     static float OutputValue = 0;
173
174     // Define index of stimulation pulse
175     static int stim_index = 0;
176
177     // Define difference between the frequency of the required stimulation pulse and the
        applied one
178     static float pulse_freq_diff = 0;
179
180     // Define state value
181     static float state_value = 0;
182
183     // Define counter for function run
184     static int timestamp = 0;
        ...
332     // Calculate current beta ARV (differential recording)
333     state_value = abs(HS1_Data_p[0] - HS1_Data_p[1]) * 10^(-12); //convert from pV to V
334
335     // Calculate Error - if SetPoint > 0.0, then normalize error with respect to SetPoint
336     if (SetPoint == 0)
337         error = state_value - SetPoint;                     //in V
338     else
339         error = ( state_value - SetPoint ) / SetPoint;          //in V
340
341     // Calculate integral term
342     ITerm += error * delta_time;
343
344     // Increment timestamp each function call and call controller when T_controller elapsed
        i.e. when timestamp == ratio_T_controller_T_s
345     if (++timestamp == ratio_T_controller_T_s)
346     {
347         // set AUX 1 output to 1
348         aux_value &= 1;
349         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
350
351         // reset timestamp counter
352         timestamp = 0;
353
354         // Calculate u(t)
355         u = Kp * ( error + 1/Ti * ITerm );
356
357         // Bound the controller output if necessary (between MinValue - MaxValue)
358         if ( u > MaxValue )
359         {
360             OutputValue = MaxValue;
361             ITerm -= error * delta_time;         // Back-calculate the integral error
362         }
363         else if( u < MinValue )
364         {
365             OutputValue = MinValue;
366             ITerm -= error * delta_time;         // Back-calculate the integral error
367         }
368         else
369             OutputValue = u;
370
```

```
371         // Determine the pulse closest to OutputValue (and its index)
372         // Set randomly the index of the pulse closest to OutputValue to be 0
373         stim_index = 0;
374 
375         // Calculate the difference between OutputValue and pulse of index stim_index
376         pulse_freq_diff = abs(pulse[stim_index] - OutputValue);
377 
378         // Pick the stimulation pulse of frequency closest to OutputValue
379         // Loop around all 16 pulses
380         for (c = 1; c < 16; c++)
381         {
382             // Check if this pulse is the closest to OuputValue
383             if ( abs(pulse[c] - OutputValue) < pulse_freq_diff)
384             {
385                 // Update the index of the closest pulse to OutputValue
386                 stim_index = c;
387 
388                 // Update the difference between OutputValue and pulse of index stim_index
389                 pulse_freq_diff = abs( pulse[c] - OutputValue );
390             }
391         }
392         // Relate the pulse index to the associated segment associated with it in the
            headstage memory
393         seg = stim_index;
394 
395         // Update stimulation pulse for channel 1
396         WRITE_REGISTER(0x9A80, 0x1000 * seg +  0x100);
397 
398         // Set AUX 1 output value to zero
399         aux_value &= ~1;
400         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
401      }
        ...
498 }
```

The only difference in terms of implementing the PI amplitude modulation control algorithm is the fact that the *MaxValue* and *MinValue* variables would be storing the maximum and minimum pulse amplitude (instead of frequency) and instead of using the *Delta_DBS_freq* variable, I would be using the variable *Delta_DBS_amp* which would hold the step in amplitude between the different stored pulses. Kindly find in the appendix (7.2) the code for implementing the PI amplitude modulation control algorithm.

In regards to the P frequency and amplitude modulation control algorithms, they are exactly similar to their associated PI control algorithm except for the fact that there's no integral term to be calculated and the fact that the *Controller Proportional Gain,* K_p has a different value for these algorithms, as summarized in table 4.1. Kindly find in appendices (7.3 and 7.4) the code for implementing the P amplitude and frequency modulation control algorithms, respectively.

Chapter 5: **Discussion and Conclusion**

During this project, we wanted to design a system that would be able to implement closed-loop deep for pre-clinical testing in animal models with parkinson's disease, that would have delays less than around $100ms$. We decided to go for the *Wireless W2100 MultiChannel Systems* device since it had the unique advantage of being the only wireless commercially available system that is able to simultaneously record and stimulate and that could potentially allow us to implement closed-loop stimulation.

I started off by implementing threshold-triggered stimulation since it can be implemented using the *MultiChannel Systems Suite* Software Package and would give us an idea of the delays I would be expecting. The *recording*, *DSP* and *stimulation delays* were very promising adding up to around $19ms$. Thus, I went on to investigate DSP-triggered stimulation.

Implementing DSP-triggered stimulation required first segmenting the Headstage memory into 16 segments and re-programming the system's firmware to enable switching between these different stimulation pulses. I then worked on coding the different closed-loop control algorithms we wanted to implement on the system, namely *on-off control*, *dual-threshold control*, *proportional amplitude and frequency modulation control* and *proportional-integral amplitude and frequency modulation control*.

Unfortunately due to the quarantine restrictions imposed due to Covid-19, I wasn't able to go to UCD to test the system's performance for these control algorithms nor quantify the delays associated with the DSP-triggered stimulation. However, as discussed in the *Future Work* chapter, I will work on that after the quarantine period is over.

Chapter 6: **Future Work**

Having successfully managed to program the W2100 system to implement the different control algorithms associated with closed-loop deep brain stimulation, hopefully after the quarantine period I will test the performance of the system for the DSP-triggered stimulation and quantify the delays associated with switching between different pulses. I will also investigate how to filter out the beta-band signal, I will check if I need to set both stimulation electrodes to stimulate or only one is enough, I will try to integrate the recorded signal from the different 8 recording electrodes into the algorithm (as opposed to using just 2 for the differential recordings), I will research the effect of implementing the control algorithms in the discrete form as opposed to continuous form where we wouldn't be limited to 16 pulse and I will try to find a way to change the 16 stored pulses online (which may require reducing the sampling frequency since I can't simultaneously record data and send stimulation pulses to the headstage). Finally, I will also work on designing a user-friendly GUI for setting up the control algorithm to be implemented on the system, in order to facilitate the use of the system for the person who will be running the experiments. Thus, hopefully by the end of June I would have the system ready to be used on pre-clinical animal trials, testing, in-vivo, the performance of the different closed-loop deep brain stimulation control algorithms.

Bibliography

[1] Ilya Adamchic, Christian Hauptmann, Utako Brigit Barnikol, Norbert Pawelczyk, Oleksandr Popovych, Thomas Theo Barnikol, Alexander Silchenko, Jens Volkmann, Günter Deuschl, Wassilios G Meissner, et al. Coordinated reset neuromodulation for parkinson's disease: proof-of-concept study. *Movement disorders*, 29(13):1679–1684, 2014.

[2] Yves Agid. Parkinson's disease: pathophysiology. *The Lancet*, 337(8753):1321–1324, 1991.

[3] Filippo Agnesi, Allison T Connolly, Kenneth B Baker, Jerrold L Vitek, and Matthew D Johnson. Deep brain stimulation imposes complex informational lesions. *PloS one*, 8(8), 2013.

[4] Chioma Anidi, Johanna J O'Day, Ross W Anderson, Muhammad Furqan Afzal, Judy Syrkin-Nikolau, Anca Velisar, and Helen M Bronte-Stewart. Neuromodulation targets pathological not physiological beta bursts during gait in parkinson's disease. *Neurobiology of disease*, 120:107–117, 2018.

[5] European Parkinson's Disease Association.

[6] Eduard Bakstein, Jonathan Burgess, Kevin Warwick, Virginie Ruiz, Tipu Aziz, and John Stein. Parkinsonian tremor identification with multiple local field potential feature classification. *Journal of neuroscience methods*, 209(2):320–330, 2012.

[7] Eduard Bakstein, Kevin Warwick, Jonathan Burgess, Øyvind Stavdahl, and Tipu Aziz. Features for detection of parkinson's disease tremor from local field potentials of the subthalamic nucleus. In *2010 IEEE 9th International Conference on Cyberntic Intelligent Systems*, pages 1–6. IEEE, 2010.

[8] Brian D Berman, Philip A Starr, William J Marks Jr, and Jill L Ostrem. Induction of bradykinesia with pallidal deep brain stimulation in patients with cranial-cervical dystonia. *Stereotactic and functional neurosurgery*, 87(1):37–44, 2009.

[9] M Beudel and P Brown. Adaptive deep brain stimulation in parkinson's disease. *Parkinsonism & related disorders*, 22:S123–S126, 2016.

[10] Merrill J Birdno and Warren M Grill. Mechanisms of deep brain stimulation in movement disorders as revealed by changes in stimulus frequency. *Neurotherapeutics*, 5(1):14–25, 2008.

[11] David T Brocker, Brandon D Swan, Dennis A Turner, Robert E Gross, Stephen B Tatter, Mandy Miller Koop, Helen Bronte-Stewart, and Warren M Grill. Improved efficacy of temporally non-regular deep brain stimulation in parkinson's disease. *Experimental neurology*, 239:60–67, 2013.

[12] Helen Bronte-Stewart, Crista Barberini, Mandy Miller Koop, Bruce C Hill, Jaimie M Henderson, and Brett Wingeier. The stn beta-band profile in parkinson's disease is stationary and shows prolonged attenuation after deep brain stimulation. *Experimental neurology*, 215(1):20–28, 2009.

[13] Peter Brown. Oscillatory nature of human basal ganglia activity: relationship to the pathophysiology of parkinson's disease. *Movement disorders: official journal of the Movement Disorder Society*, 18(4):357–363, 2003.

[14] Peter Brown, Antonio Oliviero, Paolo Mazzone, Angelo Insola, Pietro Tonali, and Vincenzo Di Lazzaro. Dopamine dependency of oscillations between subthalamic nucleus and pallidum in parkinson's disease. *Journal of Neuroscience*, 21(3):1033–1038, 2001.

[15] Carmen Camara, Pedro Isasi, Kevin Warwick, Virginie Ruiz, Tipu Aziz, John Stein, and Eduard Bakštein. Resting tremor classification and detection in parkinson's disease patients. *Biomedical Signal Processing and Control*, 16:88–97, 2015.

[16] Michael Cassidy, Paolo Mazzone, Antonio Oliviero, Angelo Insola, Pietro Tonali, Vincenzo Di Lazzaro, and Peter Brown. Movement-related changes in synchronization in the human basal ganglia. *Brain*, 125(6):1235–1246, 2002.

[17] Anna Castrioto, Eugénie Lhommée, Elena Moro, and Paul Krack. Mood and behavioural effects of subthalamic stimulation in parkinson's disease. *The Lancet Neurology*, 13(3):287–305, 2014.

[18] Anna Castrioto, Andres M. Lozano, Yu-Yan Poon, Anthony E. Lang, Melanie Fallis, and Elena Moro. Ten-Year Outcome of Subthalamic Stimulation in Parkinson Disease: A Blinded Evaluation. *Archives of Neurology*, 68(12):1550–1556, 12 2011.

[19] Chiung Chu Chen, Christof Brücke, Florian Kempf, Andreas Kupsch, Chin Song Lu, Shih Tseng Lee, Stephen Tisch, Patricia Limousin, Marwan Hariz, and Peter Brown. Deep brain stimulation of the subthalamic nucleus: a two-edged sword. *Current biology*, 16(22):R952–R953, 2006.

[20] LMF Doyle, AA Kühn, M Hariz, A Kupsch, G-H Schneider, and P Brown. Levodopa-induced modulation of subthalamic beta oscillations during self-paced movements in patients with parkinson's disease. *European Journal of Neuroscience*, 21(5):1403–1412, 2005.

[21] A Eusebio, W Thevathasan, L Doyle Gaynor, A Pogosyan, E Bye, T Foltynie, L Zrinzo, K Ashkan, T Aziz, and P Brown. Deep brain stimulation can suppress pathological synchronisation in parkinsonian patients. *Journal of Neurology, Neurosurgery & Psychiatry*, 82(5):569–573, 2011.

[22] John E Fleming, Eleanor Dunn, and Madeleine M Lowery. Simulation of closed-loop deep brain stimulation control schemes for suppression of pathological beta oscillations in parkinson's disease. *bioRxiv*, 2019.

[23] Gaia Giannicola, Sara Marceglia, Lorenzo Rossi, Simona Mrakic-Sposta, Paolo Rampini, Filippo Tamma, Filippo Cogiamanian, Sergio Barbieri, and Alberto Priori. The effects of levodopa and ongoing deep brain stimulation on subthalamic beta oscillations in parkinson's disease. *Experimental neurology*, 226(1):120–127, 2010.

[24] Christopher G Goetz, Barbara C Tilley, Stephanie R Shaftman, Glenn T Stebbins, Stanley Fahn, Pablo Martinez-Martin, Werner Poewe, Cristina Sampaio, Matthew B Stern, Richard Dodel, et al. Movement disorder society-sponsored revision of the unified parkinson's disease rating scale (mds-updrs): scale presentation and clinimetric testing results. *Movement disorders: official journal of the Movement Disorder Society*, 23(15):2129–2170, 2008.

[25] Warren M Grill, Andrea N Snyder, and Svjetlana Miocinovic. Deep brain stimulation creates an informational lesion of the stimulated nucleus. *Neuroreport*, 15(7):1137–1140, 2004.

[26] Marwan Hariz. Deep brain stimulation: new techniques. *Parkinsonism & related disorders*, 20:S192–S196, 2014.

[27] Adam O Hebb, Jun Jason Zhang, Mohammad H Mahoor, Christos Tsiokos, Charles Matlack, Howard Jay Chizeck, and Nader Pouratian. Creating the feedback loop: closed-loop neurostimulation. *Neurosurgery Clinics*, 25(1):187–204, 2014.

[28] Todd M Herrington, Jennifer J Cheng, and Emad N Eskandar. Mechanisms of deep brain stimulation. *Journal of neurophysiology*, 115(1):19–38, 2016.

[29] Julius Huebl, Christof Brücke, Gerd-Helge Schneider, Christian Blahak, Joachim K Krauss, and Andrea A Kühn. Bradykinesia induced by frequency-specific pallidal stimulation in patients with cervical and segmental dystonia. *Parkinsonism & related disorders*, 21(7):800–803, 2015.

[30] Andrea A Kühn, Florian Kempf, Christof Brücke, Louise Gaynor Doyle, Irene Martinez-Torres, Alek Pogosyan, Thomas Trottenberg, Andreas Kupsch, Gerd-Helge Schneider, Marwan I Hariz, et al. High-frequency stimulation of the subthalamic nucleus suppresses oscillatory β activity in patients with parkinson's disease in parallel with improvement in motor performance. *Journal of Neuroscience*, 28(24):6165–6173, 2008.

[31] Andrea A Kühn, Andreas Kupsch, Gerd-Helge Schneider, and Peter Brown. Reduction in subthalamic 8–35 hz oscillatory activity correlates with clinical improvement in parkinson's disease. *European Journal of Neuroscience*, 23(7):1956–1960, 2006.

[32] Andrea A Kühn, Thomas Trottenberg, Anatol Kivi, Andreas Kupsch, Gerd-Helge Schneider, and Peter Brown. The relationship between local field potential and neuronal discharge in the subthalamic nucleus of patients with parkinson's disease. *Experimental neurology*, 194(1):212–220, 2005.

[33] Andrea A Kühn, Alexander Tsui, Tipu Aziz, Nicola Ray, Christof Brücke, Andreas Kupsch, Gerd-Helge Schneider, and Peter Brown. Pathological synchronisation in the subthalamic nucleus of patients with parkinson's disease relates to both bradykinesia and rigidity. *Experimental neurology*, 215(2):380–387, 2009.

[34] Andrea A Kühn, David Williams, Andreas Kupsch, Patricia Limousin, Marwan Hariz, Gerd-Helge Schneider, Kielan Yarrow, and Peter Brown. Event-related beta desynchronization in human subthalamic nucleus correlates with motor performance. *Brain*, 127(4):735–746, 2004.

[35] Ron Levy, Peter Ashby, William D Hutchison, Anthony E Lang, Andres M Lozano, and Jonathan O Dostrovsky. Dependence of subthalamic nucleus oscillations on movement and dopamine in parkinson's disease. *Brain*, 125(6):1196–1209, 2002.

[36] Peter A LeWitt. Levodopa for the treatment of parkinson's disease. *New England Journal of Medicine*, 359(23):2468–2476, 2008.

[37] Simon Little, Martijn Beudel, Ludvic Zrinzo, Thomas Foltynie, Patricia Limousin, Marwan Hariz, Spencer Neal, Binith Cheeran, Hayriye Cagnan, James Gratwicke, et al. Bilateral adaptive deep brain stimulation is effective in parkinson's disease. *Journal of Neurology, Neurosurgery & Psychiatry*, 87(7):717–721, 2016.

[38] Simon Little and Peter Brown. What brain signals are suitable for feedback control of deep brain stimulation in parkinson's disease? *Annals of the New York Academy of Sciences*, 1265(1):9–24, 2012.

[39] Simon Little, Alex Pogosyan, Spencer Neal, Baltazar Zavala, Ludvic Zrinzo, Marwan Hariz, Thomas Foltynie, Patricia Limousin, Keyoumars Ashkan, James FitzGerald, et al. Adaptive deep brain stimulation in advanced parkinson disease. *Annals of neurology*, 74(3):449–457, 2013.

[40] Mahsa Malekmohammadi, Jeffrey Herron, Anca Velisar, Zack Blumenfeld, Megan H Trager, Howard Jay Chizeck, and Helen Brontë-Stewart. Kinematic a daptive d eep b rain s timulation for r esting t remor in p arkinson's d isease. *Movement disorders*, 31(3):426–428, 2016.

[41] Charles D Marsden and Jose A Obeso. The functions of the basal ganglia and the paradox of stereotaxic surgery in parkinson's disease. *Brain*, 117(4):877–897, 1994.

[42] Ayala Matzner, Anan Moran, Yaara Erez, Hadass Tischler, and Izhar Bar-Gad. Beta oscillations in the parkinsonian primate: Similar oscillations across different populations. *Neurobiology of disease*, 93:28–34, 2016.

[43] Cameron C McIntyre, Ashutosh Chaturvedi, Reuben R Shamir, and Scott F Lempka. Engineering the next generation of clinical deep brain stimulation technology. *Brain stimulation*, 8(1):21–26, 2015.

[44] Cameron C McIntyre and Philip J Hahn. Network perspectives on the mechanisms of deep brain stimulation. *Neurobiology of disease*, 38(3):329–337, 2010.

[45] Svjetlana Miocinovic, Suvarchala Somayajula, Shilpa Chitnis, and Jerrold L Vitek. History, applications, and mechanisms of deep brain stimulation. *JAMA neurology*, 70(2):163–171, 2013.

[46] Wolf-Julian Neumann, Katharina Degen, Gerd-Helge Schneider, Christof Brücke, Julius Huebl, Peter Brown, and Andrea A Kühn. Subthalamic synchronized oscillatory activity correlates with motor impairment in patients with parkinson's disease. *Movement Disorders*, 31(11):1748–1751, 2016.

[47] Padraig E O'Suilleabhain and Joseph Y Matsumoto. Time-frequency analysis of tremors. *Brain: a journal of neurology*, 121(11):2127–2134, 1998.

[48] Song Pan, Serdar Iplikci, Kevin Warwick, and Tipu Z Aziz. Parkinson's disease tremor classification–a comparison between support vector machines and neural networks. *Expert Systems with Applications*, 39(12):10764–10771, 2012.

[49] A Priori, G Foffani, A Pesenti, A Bianchi, V Chiesa, G Baselli, E Caputo, F Tamma, P Rampini, M Egidi, et al. Movement-related modulation of neural activity in human basal ganglia and its l-dopa dependency: recordings from deep brain stimulation electrodes in patients with parkinson's disease. *Neurological Sciences*, 23(2):s101–s102, 2002.

[50] A Priori, G Foffani, A Pesenti, F Tamma, AM Bianchi, M Pellegrini, M Locatelli, KA Moxon, and RM Villani. Rhythm-specific pharmacological modulation of subthalamic activity in parkinson's disease. *Experimental neurology*, 189(2):369–379, 2004.

[51] Alberto Priori, Guglielmo Foffani, Lorenzo Rossi, and Sara Marceglia. Adaptive deep brain stimulation (adbs) controlled by local field potential oscillations. *Experimental neurology*, 245:77–86, 2013.

[52] Emma J Quinn, Zack Blumenfeld, Anca Velisar, Mandy Miller Koop, Lauren A Shreve, Megan H Trager, Bruce C Hill, Camilla Kilbane, Jaimie M Henderson, and Helen Brontë-Stewart. Beta oscillations in freely moving parkinson's subjects are attenuated during deep brain stimulation. *Movement Disorders*, 30(13):1750–1758, 2015.

[53] NJ Ray, N Jenkinson, S Wang, P Holland, JS Brittain, C Joint, JF Stein, and T Aziz. Local field potential beta activity in the subthalamic nucleus of patients with parkinson's disease is associated with improvements in bradykinesia after dopamine and deep brain stimulation. *Experimental neurology*, 213(1):108–113, 2008.

[54] Manuela Rosa, Mattia Arlotti, Gianluca Ardolino, Filippo Cogiamanian, Sara Marceglia, Alessio Di Fonzo, Francesca Cortese, Paolo M Rampini, and Alberto Priori. Adaptive deep brain stimulation in a freely moving parkinsonian patient. *Movement Disorders*, 30(7):1003–1005, 2015.

[55] Boris Rosin, Maya Slovik, Rea Mitelman, Michal Rivlin-Etzion, Suzanne N Haber, Zvi Israel, Eilon Vaadia, and Hagai Bergman. Closed-loop deep brain stimulation is superior in ameliorating parkinsonism. *Neuron*, 72(2):370–384, 2011.

[56] P. Justin Rossi, Aysegul Gunduz, Jack Judy, Linda Wilson, Andre Machado, James J. Giordano, W. Jeff Elias, Marvin A. Rossi, Christopher L. Butson, Michael D. Fox, Cameron C. McIntyre, Nader Pouratian, Nicole C. Swann, Coralie de Hemptinne, Robert E. Gross, Howard J. Chizeck, Michele Tagliati, Andres M. Lozano, Wayne Goodman, Jean-Philippe Langevin, Ron L. Alterman, Umer Akbar, Greg A. Gerhardt, Warren M. Grill, Mark Hallett, Todd Herrington, Jeffrey Herron, Craig van Horne, Brian H. Kopell, Anthony E. Lang, Codrin Lungu, Daniel Martinez-Ramirez, Alon Y. Mogilner, Rene Molina, Enrico Opri, Kevin J. Otto, Karim G. Oweiss, Yagna Pathak, Aparna Shukla, Jonathan Shute, Sameer A. Sheth, Ludy C. Shih, G. Karl Steinke, Alexander I. Tröster, Nora Vanegas, Kareem A. Zaghloul, Leopoldo Cendejas-Zaragoza, Leonard Verhagen, Kelly D. Foote, and Michael S. Okun. Proceedings of the third annual deep brain stimulation think tank: A review of emerging issues and technologies. *Frontiers in Neuroscience*, 10:119, 2016.

[57] Andrew Sharott, Alessandro Gulberti, Simone Zittel, Adam A Tudor Jones, Ulrich Fickel, Alexander Münchau, Johannes A Köppen, Christian Gerloff, Manfred Westphal, Carsten Buhmann, et al. Activity parameters of subthalamic nucleus neurons selectively predict motor symptom severity in parkinson's disease. *Journal of Neuroscience*, 34(18):6273–6285, 2014.

[58] LA Shreve, A Velisar, NM Shanidze, BC Hill, C Kilbane, JM Henderson, et al. Incidence and modulation of resting state subthalamic nucleus beta rhythm in parkinson's disease. In *Soc Neurosci Abstr*, 2013.

[59] Aparna Wagle Shukla and Michael Scott Okun. Surgical treatment of parkinson's disease: patients, targets, devices, and approaches. *Neurotherapeutics*, 11(1):47–59, 2014.

[60] Felice T Sun and Martha J Morrell. Closed-loop neurostimulation: the clinical experience. *Neurotherapeutics*, 11(3):553–563, 2014.

[61] John A Thompson, David Lanctin, Nuri Firat Ince, and Aviva Abosch. Clinical implications of local field potentials for understanding and treating movement disorders. *Stereotactic and functional neurosurgery*, 92(4):251–263, 2014.

[62] Elina Tripoliti, Ludvic Zrinzo, Irene Martinez-Torres, Eleanor Frost, Serge Pinto, Tom Foltynie, E Holl, E Petersen, Michael Roughton, MI Hariz, et al. Effects of subthalamic stimulation on speech of consecutive patients with parkinson disease. *Neurology*, 76(1):80–86, 2011.

[63] A Velisar, J Syrkin-Nikolau, Z Blumenfeld, MH Trager, MF Afzal, V Prabhakar, and H Bronte-Stewart. Dual threshold neural closed loop deep brain stimulation in parkinson disease patients. *Brain stimulation*, 12(4):868–876, 2019.

[64] Moran Weinberger, Neil Mahant, William D Hutchison, Andres M Lozano, Elena Moro, Mojgan Hodaie, Anthony E Lang, and Jonathan O Dostrovsky. Beta oscillatory activity in the subthalamic nucleus and its relation to dopaminergic response in parkinson's disease. *Journal of neurophysiology*, 96(6):3248–3256, 2006.

[65] Diane Whitmer, Camille De Solages, Bruce C Hill, Hong Yu, Jaimie M Henderson, and Helen Bronte-Stewart. High frequency deep brain stimulation attenuates subthalamic and cortical rhythms in parkinson's disease. *Frontiers in human neuroscience*, 6:155, 2012.

[66] Brett Wingeier, Tom Tcheng, Mandy Miller Koop, Bruce C Hill, Gary Heit, and Helen M Bronte-Stewart. Intra-operative stn dbs attenuates the prominent beta rhythm in the stn in parkinson's disease. *Experimental neurology*, 197(1):244–251, 2006.

[67] Defeng Wu, Kevin Warwick, Zi Ma, Mark N Gasson, Jonathan G Burgess, Song Pan, and Tipu Z Aziz. Prediction of parkinson's disease tremor onset using a radial basis function neural network based on particle swarm optimization. *International journal of neural systems*, 20(02):109–116, 2010.

[68] S Elizabeth Zauber, Nidhi Watson, Cynthia L Comella, Roy AE Bakay, and Leo Verhagen Metman. Stimulation-induced parkinsonism after posteroventral deep brain stimulation of the globus pallidus internus for craniocervical dystonia: Case report. *Journal of neurosurgery*, 110(2):229–233, 2009.

Chapter 7: **Appendix**

7.1 Exporting Recorded Data From MC Experimenter

To export the recorded data data MC DataManager Kindly follow the steps below:

1. **Step 1:** After opening MC DataManager, click on "Refresh File List" tab, as shown in figure 7.1.

2. **Step 2:** Choose which recorded data you would like to expert, as shown in figure 7.1.

3. **Step 3:** Choose which export data type you would prefer. Fro example, to export the data in the form of a CVS file, click on "Export to ANSCII" as shown in figure 7.1.

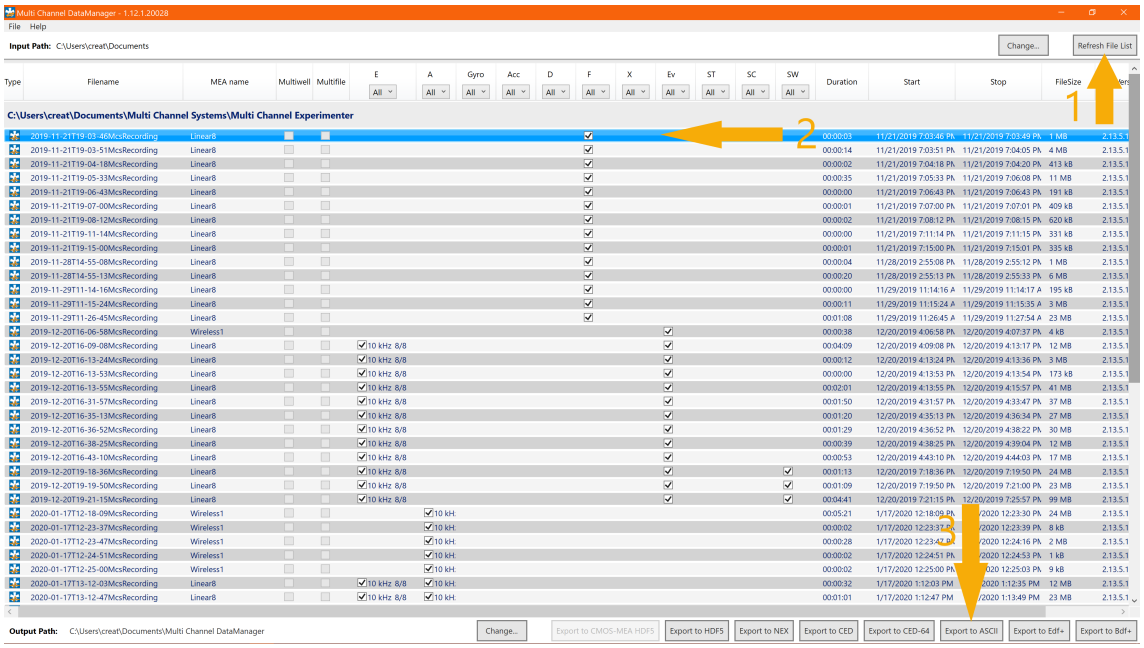


Figure 7.1: Step 3 for Uploading DSP Firmware

For further information on how to use MC DataManager kindly consult the following link `https://www.multichannelsystems.com/software/multi-channel-datamanager#docs`.

7.2 PI Amplitude Modulation Algorithm Code

```
111 interrupt void interrupt6(void)
112 {
113     // Define sampling frequency and period
114     const float f_s = 20000;
115     const float T_s = 1 / f_s;
116 
117     // Define controller call period; (20ms)
118     const float T_controller = 0.02;
119 
120     // Calculate T_controller to T_s ratio to know every how many calls of interrupt 6 we
        need to call the controller
121     const int ratio_T_controller_T_s = T_controller / T_s;
122 
123     // Define SetPoint being the target beta ARV
124     const float SetPoint = 1;                    //set SetPoint randomly to be 1V
125 
126     // Determine how aggressively the controller reacts to the current error with setting
        Proportional Gain
127     const float Kp = 0.23;                        //in AU
128 
129     // Determine how fast the controller integrates the error history by setting Integral
        Time Constant
130     const float Ti = 0.20;                        //in s
131 
132     // Define a variable that is true just the first run
133     static int first_run = 1;
134 
135     // Define the upper and lower bounds for the controller output (i.e. DBS amplitude)
136     const float MaxValue = 3000 * 10^(-6);          //in A
137     const float MinValue = 0;                       //in A
138 
139     // Define the step between pulses amplitude
140     const float delta_DBS_amp = (MaxValue - MinValue) / 15;
141 
142     // Define vector to store the amplitude of the stimulation pulses
143     static float pulse[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
144 
145     // Define dummy variable
146     static int c = 0;
147 
148     // Multiply pulse vector by delta_DBS_amp to get the vector associated with the
        amplitude of the stimulation pulses
149     if (first_run)
150     {
151         for (c = 0; c < 16; c++)
152         {
153             pulse[c] = pulse[c] * delta_DBS_amp;
154         }
155         // Not anymore the first run (thus, this if statement will be run only in the first
            call of interrupt6 function)
156         first_run = 0;
157     }
158 
159     // Define integral term
160     static float ITerm = 0;
161 
162     // Define difference in time between previous and current frames
163     const float delta_time = T_s;
164 
```

```
    // Define current and last errors
    static float error = 0;

    // Define u(t) which is calculated controller output (i.e. pulse amplitude)
    static float u = 0;

    // Define the applied bounded controller output value (i.e. applied pulse amplitude)
    static float OutputValue = 0;

    // Define index of stimulation pulse to be applied
    static int stim_index = 0;

    // Define difference between the amplitude of the required stimulation pulse and the one
    to be applied
    static float pulse_amp_diff = 0;

    // Define state value
    static float state_value = 0;

    // Define counter for function run
    static int timestamp = 0;
    ...
    // Calculate current beta ARV (differential recording)
    state_value = abs(HS1_Data_p[0] - HS1_Data_p[1]) * 10^(-12); //convert from pV to V

    // Calculate Error - if SetPoint > 0.0, then normalize error with respect to SetPoint
    if (SetPoint == 0)
        error = state_value - SetPoint;                     //in V
    else
        error = (state_value - SetPoint) / SetPoint;          //in V

    // Calculate integral term
    ITerm += error * delta_time;

    // Increment timestamp each function call and call controller when T_controller elapsed
    i.e. when timestamp == ratio_T_controller_T_s
    if (++timestamp == ratio_T_controller_T_s)
    {
        // set AUX output to 1
        aux_value &= 1;
        WRITE_REGISTER(IFB_AUX_OUT, aux_value);

        // reset timestamp counter
        timestamp=0;

        // Calculate u(t)
        u = Kp * ( error + 1/Ti * ITerm );

        // Bound the controller output if necessary (between MinValue - MaxValue)
        if ( u > MaxValue )
        {
            OutputValue = MaxValue;
            ITerm -= error * delta_time;         // Back-calculate the integral error
        }
        else if( u < MinValue )
        {
            OutputValue = MinValue;
            ITerm -= error * delta_time;         // Back-calculate the integral error
        }
        else
            OutputValue = u;

        // Determine the pulse closest to OutputValue (and its index)
        // Set randomly the index of the pulse closest to OutputValue to be 0
```

```
        stim_index = 0;

        // Calculate the difference between OutputValue and pulse of index stim_index
        pulse_amp_diff = abs(pulse[stim_index] - OutputValue);

        // Pick the stimulation pulse of amplitude closest to OutputValue
        // Loop around all 16 pulses
        for (c = 1; c < 16; c++)
        {
            // Check if this pulse is the closest to OuputValue
            if ( abs(pulse[c] - OutputValue) < pulse_amp_diff)
            {
                // Update the index of the closest pulse to OutputValue
                stim_index = c;

                // Update the difference between OutputValue and pulse of index stim_index
               pulse_amp_diff = abs(pulse[c] - OutputValue);
            }
        }

        // Relate the pulse index to the memory segment index assoicated with it
        seg = stim_index;

        // Update stimulation pulse for channel 1
        WRITE_REGISTER(0x9A80, 0x1000 * seg +  0x100);

        // Set AUX 1 output value to zero
        aux_value &= ~1;
        WRITE_REGISTER(IFB_AUX_OUT, aux_value);
     }
    ...
}
```

7.3 P Amplitude Modulation Algorithm Code

```
interrupt void interrupt6(void)
{
    // Define sampling frequency and period
    const float f_s = 20000;
    const float T_s = 1 / f_s;

    // Define controller call period; (20ms)
    const float T_controller = 0.02;

    // Calculate T_controller to T_s ratio to know every how many calls of interrupt 6 we
    need to call the controller
    const int ratio_T_controller_T_s = T_controller/T_s;

    // Define SetPoint being the target beta ARV
    const float SetPoint = 1;                    //set SetPoint randomly to be 1V

    // Determine how aggressively the controller reacts to the current error with setting
    Proportional Gain
    const float Kp = 5;                         //in AU
```

```
128
129     // Define a variable that is true just the first run
130     static int first_run = 1;
131
132     // Define the upper and lower bounds for the controller output (i.e. DBS amplitude)
133     const float MaxValue = 3000 * 10^(-6);          //in A
134     const float MinValue = 0;                       //in A
135
136     // Define the step between pulses amplitude
137     const float delta_DBS_amp = (MaxValue - MinValue) / 15;
138
139     // Define vector to store the amplitude of the stimulation pulses
140     static float pulse[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
141
142     // Define dummy variable
143     static int c = 0;
144
145     // Multiply pulse vector by delta_DBS_amp to get the vector associated with the
        amplitude of the stimulation pulses
146     if (first_run)
147     {
148         for (c = 0; c < 16; c++)
149         {
150             pulse[c] = pulse[c] * delta_DBS_amp;
151         }
152         // Not anymore the first run (thus, this if statement will be run only in the first
            call of interrupt6 function)
153         first_run = 0;
154     }
155
156     // Define current and last errors
157     static float error = 0;
158
159     // Define u(t) which is calculated controller output (i.e. pulse amplitude)
160     static float u = 0;
161
162     // Define the applied bounded controller output value (i.e. applied pulse amplitude)
163     static float OutputValue = 0;
164
165     // Define index of stimulation pulse to be applied
166     static int stim_index = 0;
167
168     // Define difference between the amplitude of the required stimulation pulse and the one
        to be applied
169     static float pulse_amp_diff = 0;
170
171     // Define state value
172     static float state_value = 0;
173
174     // Define counter for function run
175     static int timestamp = 0;
        ...
332     // Increment timestamp each function call and call controller when T_controller elapsed
        i.e. when timestamp == ratio_T_controller_T_s
333     if (++timestamp == ratio_T_controller_T_s)
334     {
335         // set AUX output to 1
336         aux_value &= 1;
337         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
338
339         // reset timestamp counter
340         timestamp=0;
341
342         // Calculate current beta ARV (differential recording)
```

```
343         state_value = abs(HS1_Data_p[0] - HS1_Data_p[1]) * 10^(-12); //convert from pV to V
344
345         // Calculate Error - if SetPoint > 0.0, then normalize error with respect to
            SetPoint
346         if (SetPoint == 0)
347             error = state_value - SetPoint;                     //in V
348         else
349             error = (state_value - SetPoint) / SetPoint;          //in V
350
351         // Calculate u(t)
352         u = Kp * error;
353
354         // Bound the controller output if necessary (between MinValue - MaxValue)
355         if ( u > MaxValue )
356             OutputValue = MaxValue;
357         else if( u < MinValue )
358             OutputValue = MinValue;
359         else
360             OutputValue = u;
361
362         // Determine the pulse closest to OutputValue (and its index)
363         // Set randomly the index of the pulse closest to OutputValue to be 0
364         stim_index = 0;
365
366         // Calculate the difference between OutputValue and pulse of index stim_index
367         pulse_amp_diff = abs(pulse[stim_index] - OutputValue);
368
369         // Pick the stimulation pulse of amplitude closest to OutputValue
370         // Loop around all 16 pulses
371         for (c = 1; c < 16; c++)
372         {
373             // Check if this pulse is the closest to OuputValue
374             if ( abs(pulse[c] - OutputValue) < pulse_amp_diff)
375             {
376                 // Update the index of the closest pulse to OutputValue
377                 stim_index = c;
378
379                 // Update the difference between OutputValue and pulse of index stim_index
380                 pulse_amp_diff = abs(pulse[c] - OutputValue);
381             }
382         }
383
384         // Relate the pulse index to the memory segment index associated with it
385         seg = stim_index;
386
387         // Update stimulation pulse for channel 1
388         WRITE_REGISTER(0x9A80, 0x1000 * seg +  0x100);
389
390         // Set AUX 1 output value to zero
391         aux_value &= ~1;
392         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
393      }
        ...
498 }
```

7.4 P Frequency Modulation Algorithm Code

```
111 interrupt void interrupt6(void)
112 {
113     // Define sampling frequency and period
114     const float f_s = 20000;
115     const float T_s = 1 / f_s;
116 
117     // Define algorithm period  (20ms)
118     const float T_controller = 0.02;
119 
120     // Calculate T_controller to T_s ratio to know every how many calls of interrupt 6 we
        need to call the controller
121     const int ratio_T_controller_T_s = T_controller / T_s;
122 
123     // Define SetPoint being the target beta ARV
124     const float SetPoint = 1;                       //set SetPoint randomly to be 1V
125 
126     // Determine how aggressively the controller reacts to the current error with setting
        Proportional Gain
127     const float Kp = 417;                          //in AU
128 
129     // Determine how fast the controller integrates the error history by setting Integral
        Time Constant
130     const float Ti = 0.20;                          //in s
131 
132     // Define a variable that is true just the first run
133     static int first_run = 1;
134 
135     // Define the upper and lower bounds for the controller output (i.e. DBS frequency)
136     const float MaxValue = 250;                     //in Hz
137     const float MinValue = 0;                       //in Hz
138 
139     // Define the step between pulses frequency
140     const float delta_DBS_freq = (MaxValue - MinValue) / 15;
141 
142     // Define vector to store the frequency of the stimulation pulses
143     static float pulse[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
144 
145     // Define dummy variable
146     static int c = 0;
147 
148     // Multiply pulse vector by delta_DBS_freq to get the vector associated with the
        frequency of the stimulation pulses
149     if (first_run)
150     {
151         for (c = 0; c < 16; c++)
152         {
153             pulse[c] = pulse[c] * delta_DBS_freq;
154         }
155         // Not anymore the first run (thus, this if statement will be run only in the first
            call of interrupt6 function)
156         first_run = 0;
157     }
158 
159     // Define current and last errors
160     static float error = 0;
161 
162     // Define u(t) which is calculated controller output (i.e. pulse frequency)
163     static float u = 0;
164 
```

```
    // Define the applied bounded controller output value (i.e. applied pulse frequency)
    static float OutputValue = 0;

    // Define index of stimulation pulse
    static int stim_index = 0;

    // Define difference between the frequency of the required stimulation pulse and the
    applied one
    static float pulse_freq_diff = 0;

    // Define state value
    static float state_value = 0;

    // Define counter for function run
    static int timestamp = 0;
    ...
    // Increment timestamp each function call and call controller when T_controller elapsed
    i.e. when timestamp == ratio_T_controller_T_s
    if (++timestamp == ratio_T_controller_T_s)
    {
        // set AUX 1 output to 1
        aux_value &= 1;
        WRITE_REGISTER(IFB_AUX_OUT, aux_value);

        // reset timestamp counter
        timestamp = 0;

        // Calculate current beta ARV (differential recording)
        state_value = abs(HS1_Data_p[0] - HS1_Data_p[1]) * 10^(-12); //convert from pV to V

        // Calculate Error - if SetPoint > 0.0, then normalize error with respect to
        SetPoint
        if (SetPoint == 0)
            error = state_value - SetPoint;                          //in V
        else
            error = ( state_value - SetPoint ) / SetPoint;           //in V

        // Calculate u(t)
        u = Kp * error;

        // Bound the controller output if necessary (between MinValue - MaxValue)
        if ( u > MaxValue )
            OutputValue = MaxValue;
        else if( u < MinValue )
            OutputValue = MinValue;
        else
            OutputValue = u;

        // Determine the pulse closest to OutputValue (and its index)
        // Set randomly the index of the pulse closest to OutputValue to be 0
        stim_index = 0;

        // Calculate the difference between OutputValue and pulse of index stim_index
        pulse_freq_diff = abs(pulse[stim_index] - OutputValue);

        // Pick the stimulation pulse of frequency closest to OutputValue
        // Loop around all 16 pulses
        for (c = 1; c < 16; c++)
        {
            // Check if this pulse is the closest to OuputValue
            if ( abs(pulse[c] - OutputValue) < pulse_freq_diff)
            {
                // Update the index of the closest pulse to OutputValue
                stim_index = c;
```

```
378
379                 // Update the difference between OutputValue and pulse of index stim_index
380                 pulse_freq_diff = abs( pulse[c] - OutputValue );
381             }
382         }
383
384         // Relate the pulse index to the associated segment associated with it in the
        headstage memory
385         seg = stim_index;
386
387         // Update stimulation pulse for channel 1
388         WRITE_REGISTER(0x9A80, 0x1000 * seg +  0x100);
389
390         // Set AUX 1 output value to zero
391         aux_value &= ~1;
392         WRITE_REGISTER(IFB_AUX_OUT, aux_value);
393      }
    ...
498 }
```