Integrating Implantable BCI Devices within BCI2000 Using a Unified Framework and Toolchain for In-Vivo Research

Dhruva Mehta
Washington University – McKelvey School of Engineering

WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Biomedical Engineering

Integrating Implantable BCI Devices within BCI2000 Using a Unified Framework and Toolchain for In-Vivo Research

by
Dhruva Mehta

A thesis presented to
the McKelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Master of Science

May 2024
St. Louis, Missouri

Table of Contents

List of Figures

Acknowledgments

This work would not have been possible without the mentorship and guidance from Dr. Peter Brunner, as well as the help and assistance from current and former members of the Brunner Lab, including Jurgen Mellinger, Markus Adamek, and William Engelhardt. I would also like to thank Dr. Alessandro Vato, Luciano Branco, the Micro-Leads team, and the Ripple team for their collaborative efforts.

Dhruva Mehta

Washington University in St. Louis
May 2024

ABSTRACT OF THE THESIS

Integrating Implantable BCI Devices within BCI2000 Using a Unified Framework and Toolchain for In-Vivo Research

by

Dhruva Mehta

Master of Science in Biomedical Engineering

Washington University in St. Louis, 2024

Professor Peter Brunner, Chair

The field of neurotechnology research has a bright and promising future as more devices are created. However, there are still many gaps in the field as the potential for neuromodulation grows. Devices such as the Micro-Leads StimZ system and the Ripple Grapevine/Summit system help bridge that gap by allowing for a broader variety of closed-loop neuromodulation experiments to be implemented thanks to their portability and versatility. Despite these devices being on the market, however, there needs to be a method to collaborate and interact with them across multiple research institutions. BCI2000 helps to address that by creating a standardized working framework for brain-computer research experiments. The implementation of the StimZ system and Ripple system within BCI2000 is just the first step towards what is possible for the field of neuromodulation.

Chapter 1

Introduction

1.1 Neuronal Electrophysiology

Neuronal electrophysiology is the study of the electrical properties of cells in the nervous system, and it forms the basis of much of the research behind brain functionality. Changes in the electrical properties of cells allow the nervous system to perform all of its many functions, from receiving sensory information, to generating a physiological response to stimuli.

The driving force behind electrophysiology is the propagation of electrical action potentials across cell membranes. Different concentrations of positively and negatively charged ions are maintained inside and outside a cell. This creates an overall large difference in voltage across the cell membrane, called the membrane potential.

Normally the difference in ion concentration would be neutralized via diffusion. However, the membrane only allows the transfer of ions through specialized channels that are opened and closed at certain voltage thresholds. The membrane potential of cells is kept quite low at -70 mV, with high concentrations of negatively charged Potassium inside the cell, and positively charged Sodium outside of the cell.

When a cell initially increases its membrane potential from other ionic membrane transfer, the voltage-gated sodium channels also begin to open, further increasing the membrane potential. This rapid depolarization of the cell propagates across the whole cell membrane by causing more sodium channels to be opened down the length of the cell. Through these action potentials, the central nervous system interacts with the body through the secretion of hormones and contracting of muscles.

There are many ways to actually study the electrophysiology of the nervous system. However the fundamental mechanism behind all of them is to acquire and measure the electrical activity of the nervous system to make scientific breakthroughs. This is done through intra- and extracellular recordings.

Originally, intracellular recordings were done through the insertion of a microelectrode into a cell's interior. These microelectrodes consisted of a glass pipette filled with intracellular fluid and a metal wire inside the pipette to connect the solution to a recording device. However, there is an alternative called the patch-clamp technique. Rather than impaling the cell, patch-clamp micropipettes are placed on a "patch" of the cell membrane. By using suction, the patch of the membrane is drawn into the tip of the pipette, allowing the pipette to be attached to the cell. Using more suction will cause the patch to break off into the pipette, leaving the pipette connected to the cell's interior. Other methods of patch clamping, such as perforated, loose, inside-out, and outside-out patches, allow specific control over which properties of the cell and cell membrane are studied. Other clamp methods for intracellular recordings include the voltage clamp method and the current clamp method. The voltage clamp method maintains the membrane voltage at a specific value through altering currents. This allows researchers to study how changes in membrane potential affect the cell and cell membrane. The current clamp method allows researchers to inject positive or negative current into the cell and observe the cell and cell membrane's response to the stimuli.

While intracellular recordings are good at observing the electrophysiology of a single cell, extracellular recordings are able to acquire data on multiple cells, as well as extracellular fluid. Recordings are done by measuring the changes in the potential of cells surrounding a microelectrode. It is difficult to distinguish individual cells when using a full microelectrode array, because the position and size of each microelectrode can change the measurement. As a result, a process called spike sorting is used to discern the activity of individual neurons from the overall larger electrical waveform data.

1.2 Neuromodulation and Devices

Neuromodulation is defined as "the alteration of nerve activity through targeted delivery of a stimulus, such as electrical stimulation or chemical agents, to specific neurological sites in the body." [3]. Both intracellular and extracellular methods form the backbone of neuromodulation research and therapies. The main form of distinction in neuromodulation research is between open-loop and closed-loop systems.

Open-loop neuromodulation systems deliver neurostimulation at consistent time periods with a predetermined schedule. After the neurostimulation, measurements are taken on the subject to characterize the response to the stimulus. These measurements are then analyzed at a later point in time by the investigator [4]. However, many times the delivery of neurostimulation may need to be contingent on specific timings or physiological states. This is where closed-loop systems can be used. In a closed-loop system, the researcher will set specific conditions for the delivery of stimulation. Then, an automated program will evaluate the measurements and current states of the subject before determining if the conditions to deliver stimulation are fulfilled again. Open-loop systems are good for testing novel therapies, as there are not many known results to actually create a set of rules and conditions to stimulate the subject. Stimulating agnostic to specific states of the subject allows a variety of data to be acquired for future research. On the other hand, closed-loop systems are useful for implementing known solutions and therapies, as well as refining therapeutic processes. This is because an automated process will deliver optimal stimulation when the conditions are most beneficial to the subject and iterate based on the data acquired.

There are also a plethora of possible research applications for closed-loop neuromodulation, many of which require specific parameters for devices. As a result there is no one size fits all solution and there are a wide variety of devices on the market. For example, the Cortec Brain Interchange is a fully implantable system for long-term measurement and electrical stimulation of both the central and peripheral nervous system. Its portability is contrasted with the Neuro-Omega, which is a larger device used for closed-loop stimulation during deep brain stimulation procedures.

However, despite the many options available on the market, there is still a severe gap in the field. Considering the density of technology surrounding closed-loop neuromodulation, many of the current solutions are not portable. This sometimes severely limits the ability to acquire research data. As a result, portable solutions are in a high demand. The two devices discussed in this paper, the Micro-Leads Stim-Z system and the Ripple Grapevine/Summit, fill this gap in the application of neuromodulation devices.

The Micro-Leads Stim-Z System

The Micro-Leads Stim-Z system consists of multiple components:

- One or more wireless implantable devices

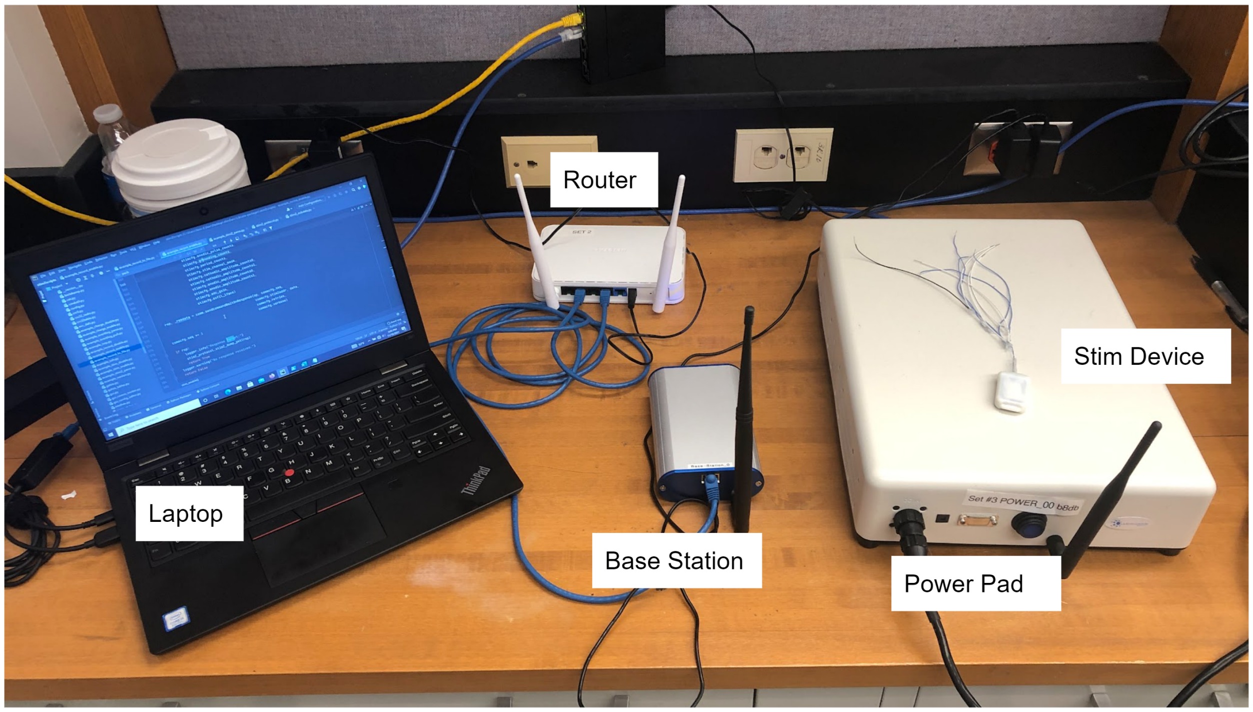


Figure 1.1: An example setup for the Micro-Leads System with the 5 main components

- One or more wireless charging power pads for the implantable devices
- One base station that bridges Stim-Z wireless links and ethernet
- One router for creating an ethernet sub-network for base station communication
- One computer for Stim-Z system command

One of the interesting things about the Stim-Z system is that it can support multiple implantable devices and charging pads. Each base station can be connected to 7 power pads and 7 implants at once, as well as monitor and charge all of them at the same time. However, it can only stream recording data from one implant at a time, and each device has to be on a unique frequency channel, of which there are 14 channels in total.

The Ripple Grapevine and Summit are larger in comparison to the Stim-Z system's implantable device, but with that size came more capabilities. Both the Grapevine and Summit support up to 512 channels of stimulation and recording as well as support for interfacing directly with third-party devices via different front-end modules. It also supports 6 different recording modes depending on the quality and speed of data the user requires.

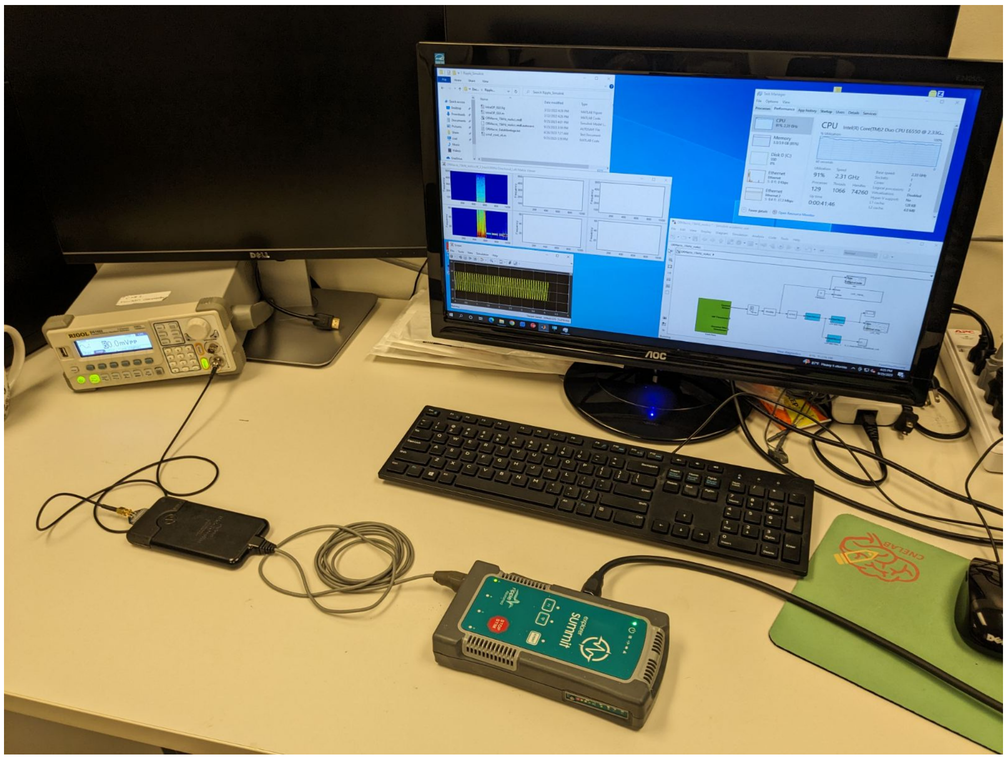


Figure 1.2: An example setup for the Ripple System, courtesy of Luciano Branco

1.3 Communication with Devices

As part of the conditions to fulfill grant or funding requirements, companies usually develop a piece of software that can be used to communicate and use the device. As a result, usually each device will be shipped with a graphical user interface (GUI) or an application programming interface (API). Most companies developing hardware for brain computer interface research applications tend to focus on hardware. Thus, the quality of the software can vary drastically depending on factors like previous devices made and holistic software suites. Furthermore, there is no standardized way to create the software, as well as industry-standard communication protocol. Depending on the company's expertise, the communication can be facilitated at a high or low level of programming abstraction. For example, Micro-Leads has a Python API that utilizes UDP and TCP communication protocols, but its main form of actually communicating between devices is via a radio link. Ripple, on the other hand, has a Python API that is heavily abstracted for the user, so as to not introduce complications and confusion. This can lead to researchers having to learn software and communication protocols on a device-by-device basis. On the other hand, researchers may be more inclined to stick with only one company's products. However, this is inherently limiting options they would have for their research, as well as for options for collaboration. When communication protocols are device-specific, it makes reproducing experiments and collaborating with other research institutions difficult to facilitate.

BCI2000 is built to address the issues that plague the current state of neurotechnology research. BCI2000 is an open-source general-purpose software platform for brain computer interface research. It allows for a standardized way to create and run experiments, abstracted from specific device protocols and software. This then allows it to be used across multiple research institutions by lowering the barrier to collaboration through removing the constraint of device-specificity. It also supports a plethora of features such as real-time visualization, closed-loop experimentation, pre- and post-processing chains, and more.

Part of the way that BCI2000 fills this gap is through constant development over the past 20 years. This means that as new devices come out, they need to be integrated to be used with BCI2000 in a way that facilitates communication and use as close to the native software capabilities as possible. BCI2000 uses the APIs associated with each device to allow for consistent and working software to communicate with the device. However, due to the robust toolchains and previous development for BCI2000, many times the capabilities of the device are improved from normal use with their original software suite. Furthermore, it allows for ease of use with many previously integrated periphery accessories and devices, allowing researchers to be comfortable with diving deep into the software's capabilities. This paper discusses the integration of the Micro-Leads Stim-Z system and Ripple Grapevine/Summit with BCI2000.

Chapter 2

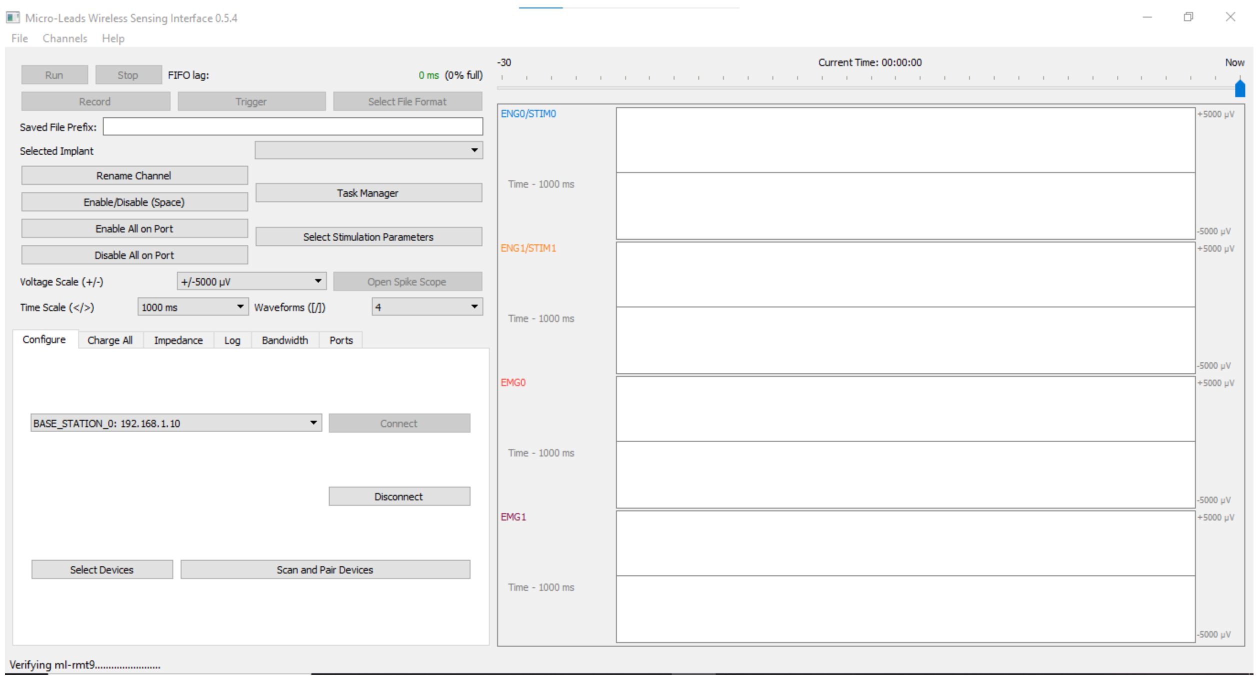
Methods

2.1 Micro-Leads Stim-Z System

Integrating the Micro-Leads Stim-Z System with BCI2000 initially was thought to be a straightforward exercise in creating a module to allow the system to be used as the signal source for an experiment. However, the system quickly had many complications and difficulties that needed to be addressed. Originally, the Stim-Z System was meant to be used with a graphical user interface (GUI) that was provided by Micro-Leads themselves. A researcher would be able to control many facets about the system, such as maintaining a connection between the different hardware, charging the stimulation device, and initiating a recording trial.

The user would be able to set their parameters, watch a live visualization of the incoming data stream, and then be able to look at the saved data from a file. While this was adequate for device-specific software, there was no real way to coordinate its use within a larger experimental procedure. Furthermore, there were a plethora of communication issues that resulted in device malfunction and data loss.

To this end, the Stim-Z system had a Python-based application programming interface to establish external communication with the various hardware devices in the system without requiring use of the GUI. However, the Python API also had a slew of problems. The main concern with the API was that its core function to acquire recording or stimulation data wrote the acquired data to a file. This meant that anyone using the API wasn't able to view the data in real-time and had to wait until the experimental trial was over to be able to check if there were any issues.



FIFO lag: 0 ms (0% full)

Saved File Prefix:

Selected Implant

Voltage Scale (+/-) +/-5000 µV

Time Scale (</>) 1000 ms

Waveforms ([/]) 4

BASE_STATION_0: 192.168.1.10

Current Time: 00:00:00

Verifying ml-rmt9.......................

Figure 2.1: An image of the GUI provided by Micro-Leads for use with the Stim-Z system

So on one hand, there was an inconsistent GUI that could not be used effectively in a larger closed-loop experimental procedure, and on the other hand there was a Python API that could not visualize data in real-time. The goal of using BCI2000 was to resolve these issues. As discussed previously, the API is the main method of integration with BCI2000. However, BCI2000 is written in C++ while the Stim-Z API is written in Python. This meant that the API in its current form was incompatible with BCI2000 without a workaround, which posed the first integration problem.

At first the solution was to figure out how to intercept the data stream that was being saved to a file via the python API and send it to BCI2000 instead, where it could be visualized, processed, and used in the larger ecosystem. Although Python and C++ are different languages, Python is actually built as an extension of C++. As a result, conceptually, it was possible to create a C++ wrapper around the Python API to allow for BCI2000 to execute Python commands within its normal operations.

However, testing showed that there were latency concerns with using a wrapper around the Python API. Timing is of the utmost importance for data acquisition, especially with closed-loop neuromodulation. If data takes too long to be acquired, it can severely limit the overall quality of the experimental results, if not outright make the experimental procedure unable to be run in the first place. On top of the latency concerns, there were also concerns with the Python API's ability to acquire consistent, high-quality data with low loss.

As a result, a new solution had to be developed. During the first development process to use the Python API, there was an interesting learning. While the API was written in Python, the GUI that Micro-Leads shipped with the Stim-Z system was written in C++. Ostensibly, there had to be some C++ API that the GUI used to communicate with the device in a similar way to the Python API. However, when approached with a request to access the API, Micro-Leads could not provide it, as the API had been lost from transitions in projects and former employees moving on to other companies.

While this usually might spell certain doom for a project, it was not a complete dead end. Even though there was no way to access the original C++ API, theoretically there was a way to recreate it from scratch. Since the GUI and Python API were provided, it was possible to reverse-engineer the API by observing the communication protocols in action on a networking level. From these observations, the original functionality of the API could be recreated within a BCI2000 source module. Furthermore, since BCI2000 would be establishing a connection with the system without the need for an API, there would be lowered latency from the removal of a "middleman".

To recreate the API, several steps needed to be taken. First, there needed to be functions that established the main networking and port connections that allowed for communication between the hardware of the Stim-Z system and BCI2000. Then, functions that requested statements of health from the hardware needed to be created. Next, functions that managed the acquisition of recording and stimulation data from the main stimulation device were established. Finally, the data packets acquired needed to be decoded and integrated with BCI2000 to be visualized.

2.1.1 Communication Protocol

The Stim-Z system has a few layers for its communication protocol. At the center of the entire system is the base station, as the base station initiates all wireless communication during normal operation. The base station receives commands from the laptop or computer serving as a command center in the form of Stim-Z protocol packets. It then relays the

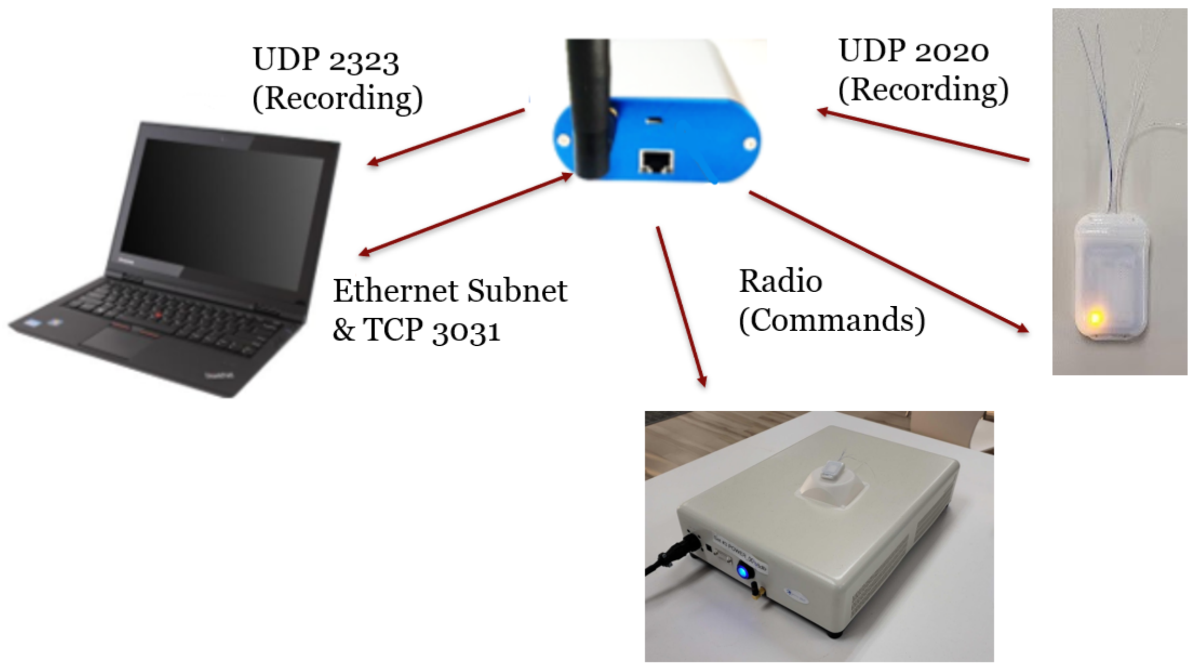


Figure 2.2: A diagram demonstrating the pathways for communication and data exchange within the Stim-Z system.

packet to the specific device addressed in the packet over a radio link. Finally, there is a separate command interface specifically for streaming data connection requests and delivery.

The command interface for the Stim-Z command relay is maintained through a Transmission Control Protocol (TCP) connection. TCP is one of the main communication protocols for sending and receiving messages over a network. When using TCP, the two devices first establish a connection, then send data packets, and finally close the connection. To establish a connection, the devices initiate a three-way handshake. First, the sender sends a packet request to synchronize. The receiver then sends an acknowledgment packet back. Finally, the sender also sends an acknowledgement packet to the receiver. Now the two devices can send and receive data packets. Data packets follow a specific packet format that includes the source and destination, data, an acknowledgement number, and more. When the data is received, the acknowledgement is incremented by the length of the total data. This allows for the two devices to keep track of when things may go wrong during communication, such as data packet loss or doubling.

After the command packet is received by the base-station, it will analyze the data packet and then forward the command to the respective device that was stated in the data via a radio link. The base-station has two radio modes to communicate with devices. The first mode is a low-power mode that is used for sending commands to receive information and facilitate basic functions such as charging and powering on or off. The second mode is a high-speed mode that allows for the streaming of data from an implantable device. These two modes have to be manually set via a command to the base station. For example, when powering a device on, the base station's radio mode needed to be in low-power mode, while when initiating the recording of streaming data, both the base station and the implant needed to be in high-speed mode. Furthermore, the high-speed modulation mode does not allow for normal commands for statements of health or charging to be sent. As a result, another aspect of the API was making sure that the radio modes for all of the devices were consistent with the command that was being sent. One other aspect of the radio link is that it is an unreliable connection. If a command packet is dropped in the radio link between the base station and the target device, a new command needs to be sent. As a result, another layer to the API needed to be implemented via a retry scheme to make sure that commands were executed properly in order.

Finally, the last aspect for the communication protocol is the command interface for streaming data from the implantable device. This command interface is facilitated by universal diagram protocol or UDP. Like TCP, UDP is a communication protocol used for sending and receiving data packets. UDP is much faster in comparison to TCP and other communication protocols, making it very good for applications with time and latency concerns, such as real-time data acquisition. However, it comes with a drawback. As discussed previously, TCP keeps track of when data packets are lost, doubled, or out of order and is able to correct them. UDP, however, does not have these features. UDP only has a checksum that can determine if a data packet was corrupted in some way, but not how it was corrupted. Furthermore, it can't reconstruct the data or fix order like TCP can with sequence numbers. The base station listens on one UDP for a request for a streaming data connection. It will then stream the recording data packets back to the sender on another, different UDP port.

2.1.2 API Reconstruction

After the communication protocols were configured to be used in a BCI2000 source module, the next step was to create separate function to handle specific commands between BCI2000 and the devices. This was where most of the functionality of an API would be found, as, normally, one would just call a function from the API to handle communication. Without the C++ API, these functions needed to be recreated. Since these functions were essentially creating data packets to be sent via the communication protocols discussed previously, the main concern is figuring out how these data packets were created.

Wireshark is a program that allows users to observe connections between devices formed from communication protocols. As a result, a user can watch connections and then analyze the data packets that are being sent. Wireshark was able to save the entire Stim-Z command relay between the native GUI, base-station, and implantable device. The data packets were then matched up with the specific commands based on the provided Python API documentation by analyzing the Stim-Z Protocol Packet Header. The individual packet type and data payload were different depending on the value of the header's OP byte. As a result, commands to receive statements of health could be distinguished from commands to initiate recording, as an example.

The next step was to use this information to create initial functions to poll devices for information on their health, such as temperature, battery voltage, and current. These statements of health would only be acquired if the device is powered on and has enough charge, so they were used as sanity checks when BCI2000 was initialized to make sure that all of the hardware is powered on and communication was established properly. Furthermore, statements of health were required at regular intervals to make sure that functionality like recharging the implant was maintained.

Then, functions to enable, initiate, and disable recording were implemented following the matching data packet information. Due to the required layers in the communication protocol, initiating recording of the data stream required multiple functions in conjunction. First, the base station needed to change its radio mode to the low-power mode. After this, it could send a command to the implant to change its radio mode to high-speed. After this, the base station would change its radio mode to high-speed as well. Then, the stimulation and

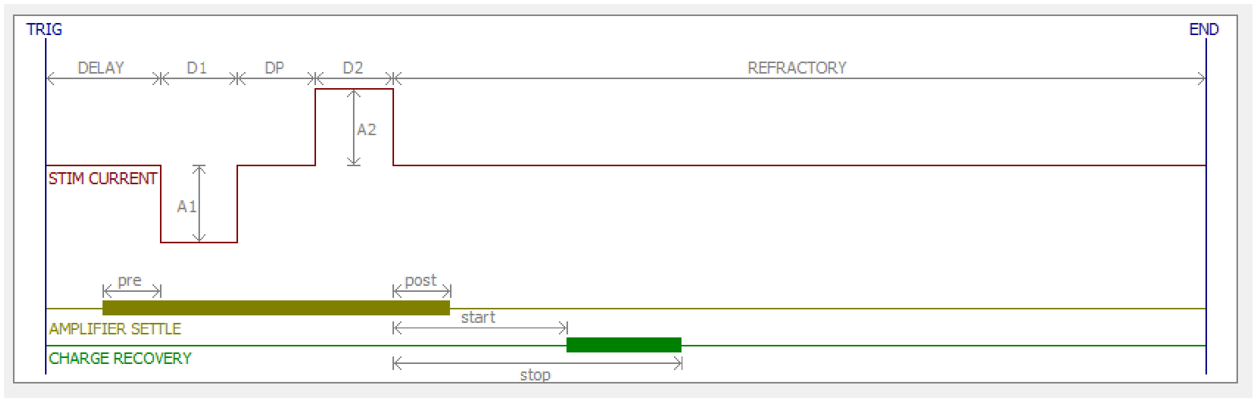


Figure 2.3: An example waveform diagram for the Stim-Z system to be emulated within BCI2000

power supply in the implant is turned on. Finally, the base station can start streaming the recording data from the implant.

Now that a way to access the recording stream had been established within BCI2000, it was necessary to visualize the data. Part of the benefits of data packets is the ability to send compressed data quickly. However, this means that after the data packet is received, it needs to be decoded. In this case, the incoming data was purely packed byte code, and the original API provided a rough scheme of how the data was packed. Therefore, one could reverse the packing steps to unpack the data and output it. However, the scheme that the original API provided wasn't completely accurate. With the given packet structure, the data was conceived to be 160, 6 sample tuples. However, in reality, the data packet was 80, 6 sample tuples instead. This meant that initially half the data received was blank noise. After the decoding errors were corrected, the last step was to implement stimulation to match the parameters set by the original API. One of the main goals of implementing a source module in BCI2000 is to have the device maintain similar functionality to their native software or platform. This meant adjusting the normal BCI2000 parameters to the stimulation waveform of the implantable device, as shown here.

2.2 Ripple System

In comparison to the Stim-Z system, the Ripple Grapevine/Summit Neural Interface Processor was more straightforward. The Ripple Grapevine/Summit also had its own API

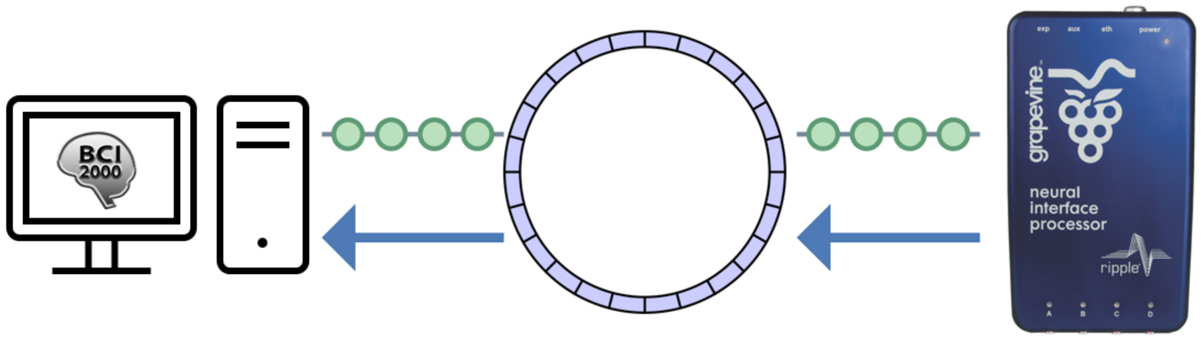


Figure 2.4: An image showing the circular buffer mechanism for Ripple hardware

and native GUI software. Like the Micro-leads Stim-Z system, the API was written in Python. However, at the time of implementation, version 1.0 of BCPy2000 had been released. BCPy2000 is a Python version of BCI2000 that has all of its original functionality, but in Python. This meant that the API would be able to work seamlessly with a version of BCI2000 without any latency concerns. Since BCPy2000 had not had a source module implemented at that time, this was the perfect opportunity to test it in a real-world setting.

We were unable to acquire a loaner device to test with during the development, so I had to rely on our collaborators to borrow their devices via remote access software. During the development process, the main concerns were maintaining the strict timing and latency requirements that the device held due to its data acquisition process. Instead of constantly sending a stream of data to an access point, the device outputs data to a rolling buffer with a specific window size. Users acquire data by polling the buffer with a timestamp and the number of data points desired.

Since this was a rolling buffer, after a certain amount of time, data would be discarded. This meant that if there was a timing issue between the timestamp requested and the current time on the device, there would be problems with the constant data stream. If the time between data acquisition blocks was too long, eventually the data buffer would not have data at the timestamp requested. If the time between data acquisition was too short, eventually there would not be enough data in the buffer when requested, causing the program to crash if not handled properly.

Chapter 3

Results

3.1 Micro-leads Stim-Z System

Leads for each channel were not explicitly labeled, so manual testing was done to determine which leads corresponded to each channel. Recording output was verified using a g.tec sine signal generator. Stimulation output was verified using a Tektronix oscilloscope. Both types of output were plotted in MATLAB to make sure that frequency and amplitude matched input parameters.

There were three main data acquisition methods tested for the device: Native GUI, Python API, and BCI2000 source module. Average round-trip latency were calculated for each method after data acquisition, as seen in Figure 3.5.

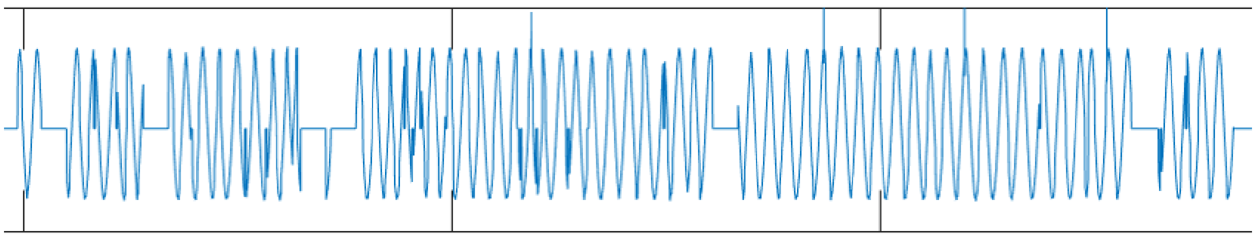


Figure 3.1: An example recording data output when using the GUI to communicate with the device

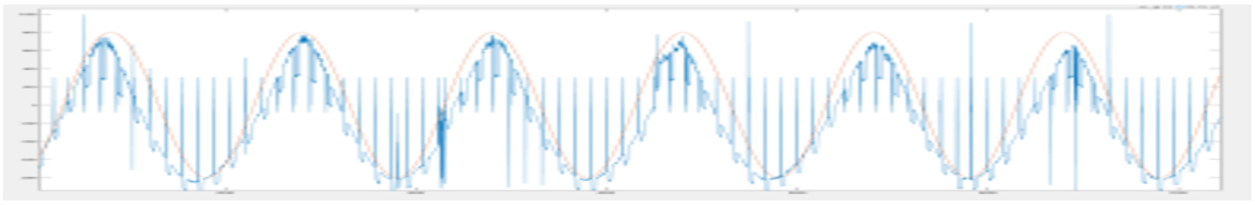


Figure 3.2: An example recording data output when using the Python API to communicate with the device

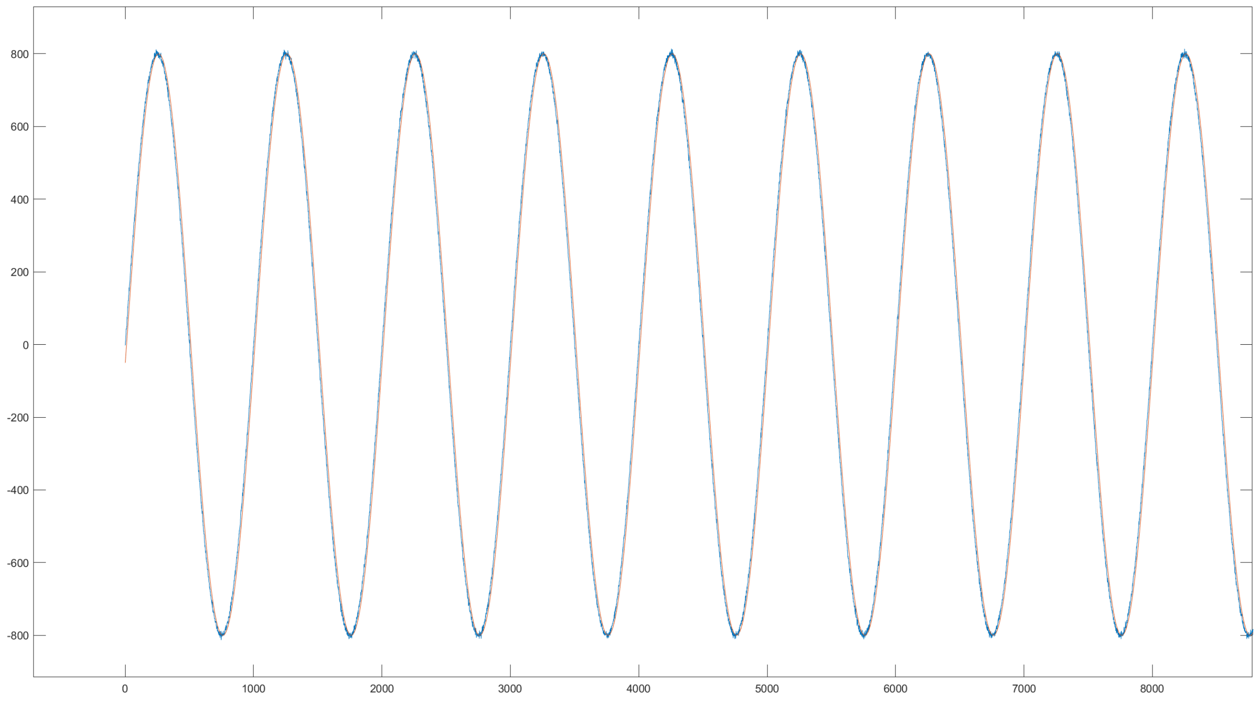


Figure 3.3: An example recording data output when using BCI2000 to communicate with the device

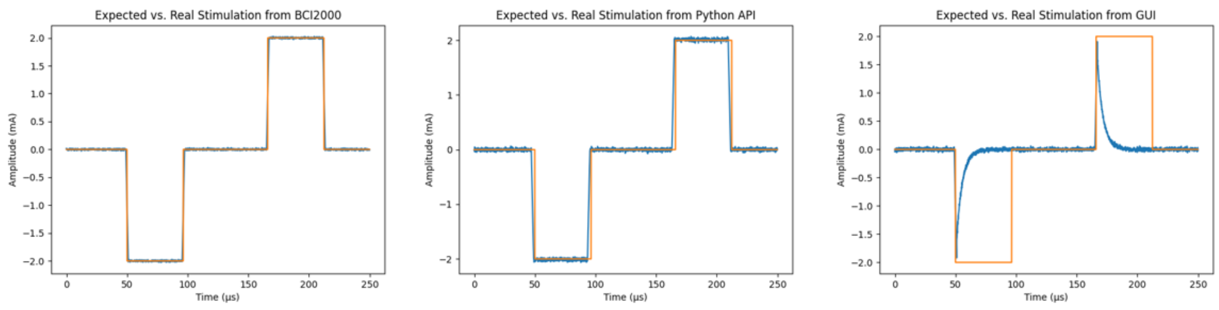


Figure 3.4: Graphs depicting the expected vs. real stimulation output of the device based on the method of interfacing.

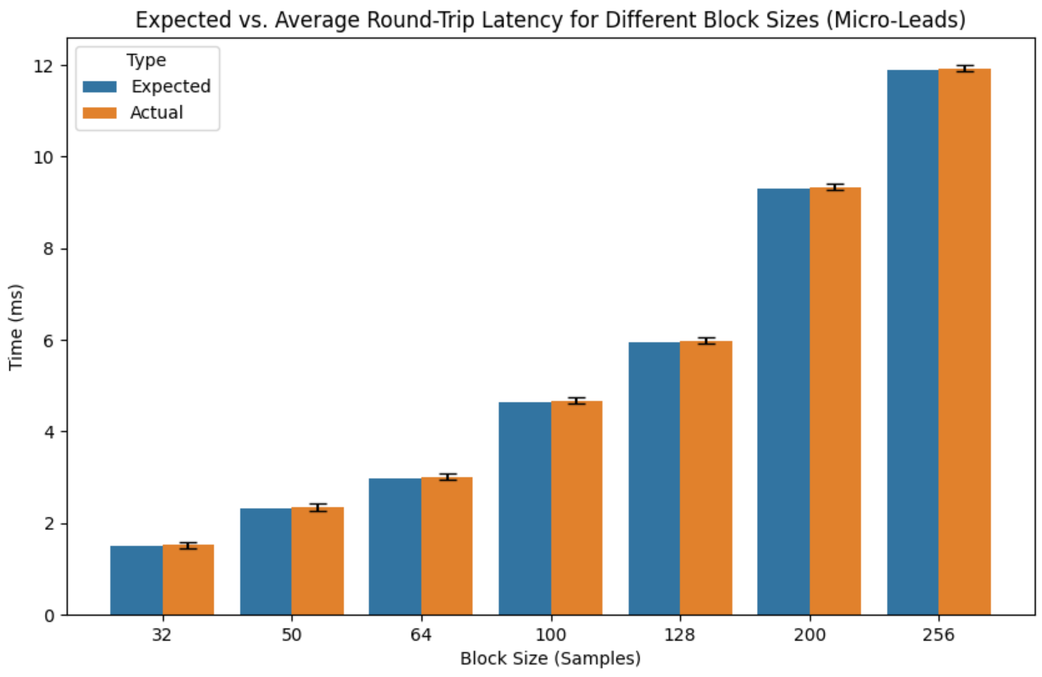


Figure 3.5: A graph depicting the average latency between blocks based on block size for the Stim-Z System

3.2 Ripple System

The main goal for the integration of the Ripple system was to match the previous method of data acquisition through MATLAB. The following figures show the average round-trip latency for several block sizes and two different front-end peripherals: 30kHz and 15kHz.

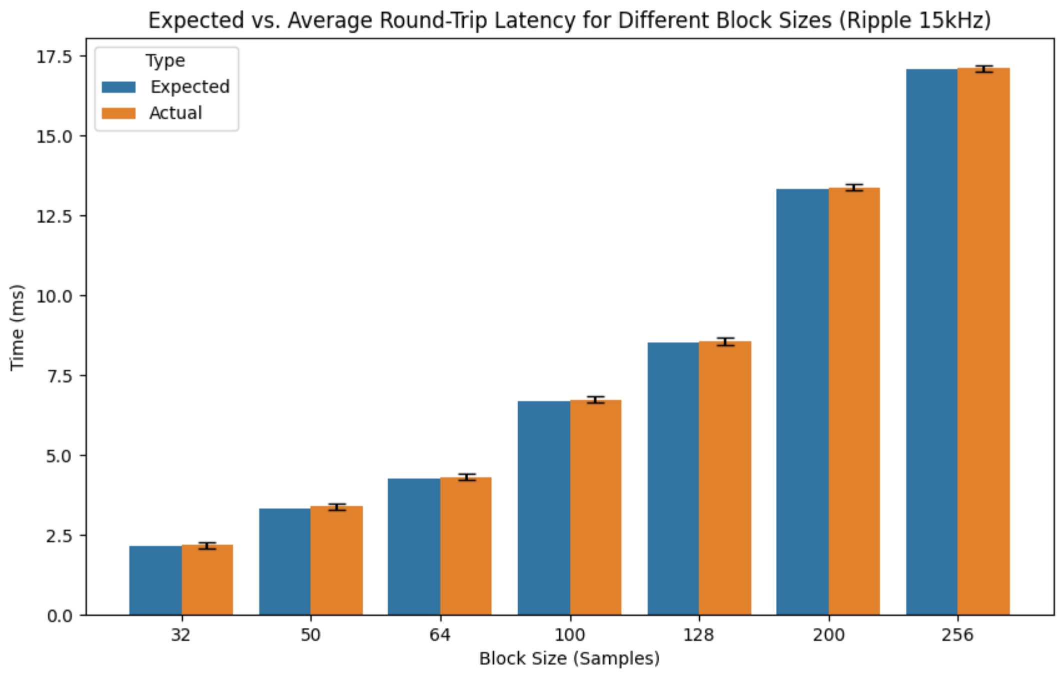


Figure 3.6: A graph depicting the average latency between blocks based on block size for the Ripple system at a sampling rate of 15kHz.

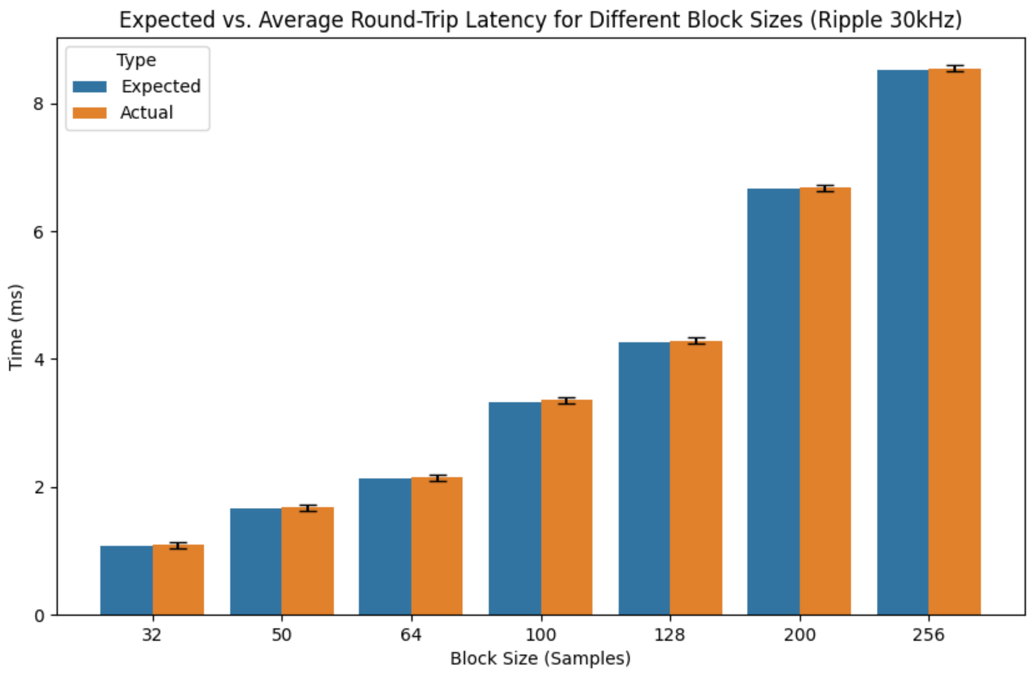


Figure 3.7: A graph depicting the average latency between blocks based on block size for the Ripple system at a sampling rate of 30kHz.

Chapter 4

Discussion

4.1 Micro-Leads Stim-Z System

There are many interesting aspects to the results from implementing the Micro-Leads Stim-Z System. One of the main points of comparison was between the results from interfacing with the device with the native GUI, the Python API, and BCI2000. From Figures 3.1 and 3.2, we can see how interfacing with the GUI and the Python API gave inconsistent results for recording. From the multiple trials that I ran, there was no result that had a consistently clear signal generated from the g.Tec signal generator. There was a difference between the two, however. The native GUI would have some parts of the signal generated cleanly, with much of the signal being lost. The Python API, on the other hand, consistently had an error in its data that caused the spiking noise behavior. Figure 3.3 shows how BCI2000 relayed a clear and consistent signal generated after loading the data acquired into MATLAB.

The results from comparing the stimulation output between the three methods of communicating with the Stim-Z system was also quite interesting. As shown in Figure 3.4, both BCI2000 and the Python API generated a relatively consistent stimulation waveform. However, the GUI would many times output the waveform depicted in the figure. Instead of stimulating for the entire stimulation period, it would stimulate as a spike before petering off in an exponential decay.

Finally, the results comparing the expected vs. average block latency were as expected. As the block size increases, the average time between blocks will also increase. Furthermore, the realized average round-trip latency was slightly higher than expected, attributing to some of the delay in data acquisition. Lastly, the standard deviation for each block size remained constant, and the percent standard deviation decreased as block size increased.

Overall, it was interesting to see that the BCI2000 implementation of the Stim-Z system may have actually improved the devices capabilities.

4.2 Ripple System

The Ripple system also had interesting results. The focus for the Ripple devices was on the implementation of its recording capabilities. However, with that, there were 6 different methods of recording. I compared two of the methods that depended on differing front-end attachments for the device. The Macro front-end attachment samples at a rate of 15000 Hz, while the Micro front-end attachment samples at a rate of 30000Hz. The sampling rate of the device can also go down to as low as 2000 Hz, making consistent recording important with varying sampling rates.

Average round-trip latency for both front-ends were derived and analyzed. The average round-trip latency for both front-ends stayed at a relatively constant percent at around 1% instead of constant in total time. This means that as block size increased, so did the average round-trip latency which is unusual from expected behavior. As seen with the Stim-Z system, average round-trip latency should generally stay constant with block size. As a result, it is generally encouraged to increase block-size to make sure that actual round-trip latency does not deviate too far from the expected round-trip latency. However, the Ripple devices performed well even under extremely low block sizes. This may indicate some other issue at hand, such as performance issues with the computer running BCI2000 or some larger issue with BCPy2000.

Chapter 5

Conclusion

I would now like to take the time to discuss my personal learning from the development of this thesis as well as to briefly state what I implemented. When first undertaking this project, I only had experience in software engineering. However, I quickly realized how multidisciplinary brain-computer interfacing research, and by extension, biomedical engineering research was. I have learned many important lessons, from engineering skills regarding hardware and networking, to personal skills in problem solving and reaching out to my peers. Overall, it has been an incredible experience, and I can't thank those who have helped me enough.

During the course of this thesis, I integrated the Micro-Leads Stim-Z system within BCI2000. To do this, I refactored the Python API into C++ while also correcting mistakes regarding its data acquisition and decoding process. I then conducted testing on the entire system to ensure data acquired was valid and within reasonable round-trip latency constraints. I then created documentation for the implementation in the form of a wiki page and tutorial video. For the Ripple Grapevine/Summit system, I created the first source module for the modern version of BCPy2000. I also verified its capabilities with the assistance of Luciano Branco. I have also created documentation for the implementation in the form of a wiki page and tutorial video, as well as assisted with its integration in the MATLAB Simulink toolchain pipeline.

Overall, the possibilities for closed-loop neuromodulation have improved with the integration of the Micro-Leads Stim-Z system and Ripple Grapevine/Summit with BCI2000. Experimental paradigms that rely on portability and size will now be able to be realized. Furthermore, they will be able to be replicated across multiple institutions through the standardized framework of BCI2000, allowing for unparalleled levels of collaboration. As more devices are created and integrated in BCI2000, more steps can be taken to uncharted territory in neuromodulation research.

References

[1] P. Ghasemi, T. Sahraee, and A. Mohammadi. Closed- and open-loop deep brain stimulation: Methods, challenges, current and future aspects. *Journal of Biomedical Physics and Engineering*, 8(2), Jul 2018.

[2] T. Haeusermann, C. R. Lechner, K. C. Fong, A. Bernstein Sideman, A. Jaworska, W. Chiong, and D. Dohan. Closed-loop neuromodulation and self-perception in clinical treatment of refractory epilepsy. *AJOB Neuroscience*, 14(1):32–44, Sep 2021.

[3] I. N. Society. Neuromodulation, or neuromodulatory effect. `https://www.neuromodulation.com/`. Accessed: 2024-05-06.

[4] S. Zanos. Closed-loop neuromodulation in physiological and translational research. *Cold Spring Harbor Perspectives in Medicine*, 9(11), Dec 2018.